

WebSphere Application Server V6.1:

System Management and Configuration

**Learn about WebSphere Application
Server**

**Configure and administer a
WebSphere system**

Deploy applications



**Carla Sadtler, Fabio Albertoni,
Bernardo Fagalde, Thiago
Kleinubing, Henrik Sjostrand, Ken
Worland, Lars Bek Laursen, Martin
Phillips, Martin Smithson,
Kwan-Ming Wan**

ibm.com/redbooks

Redbooks

Archived



International Technical Support Organization

**WebSphere Application Server V6.1:
System Management and Configuration**

November 2006

Archived

SG24-7304-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xv.

First Edition (November 2006)

This edition applies to IBM WebSphere Application Server V6.1, IBM WebSphere Application Server Network Deployment V6.1, and IBM WebSphere Application Server for z/OS V6.1.

© Copyright International Business Machines Corporation 2006. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	xv
Trademarks	xvi
Preface	xvii
The team that wrote this redbook	xvii
Become a published author	xxi
Comments welcome	xxii
Part 1. The basics	1
Chapter 1. WebSphere Application Server	3
1.1 Product overview	4
1.2 WebSphere Application Server	5
1.3 Packaging	7
1.4 Supported platforms and software	10
1.4.1 Operating systems	10
1.4.2 Web servers	11
1.4.3 Database servers	12
1.4.4 Directory servers	12
Chapter 2. System management: A technical overview	15
2.1 System management overview	16
2.1.1 System management in a stand-alone server environment	16
2.1.2 System management in a distributed server environment	17
2.2 Java Management Extensions (JMX)	19
2.2.1 JMX architecture	19
2.2.2 JMX distributed administration	22
2.2.3 JMX MBeans	24
2.2.4 JMX usage scenarios	24
2.2.5 J2EE management	25
2.3 Distributed administration	26
2.3.1 Distributed process discovery	28
2.3.2 Centralized changes to configuration and application data	31
2.3.3 File synchronization	32
2.4 Configuration and application data repository	39
2.4.1 Repository directory structure	39
2.4.2 Variable scoped files	42
2.4.3 Application data files	42

Chapter 3. Getting started with profiles	47
3.1 Understanding profiles	48
3.1.1 Types of profiles	50
3.1.2 Directory structure and default profiles	51
3.2 Building a system using profiles	54
3.2.1 Stand-alone server environment	55
3.2.2 Distributed server environment	55
3.3 Creating profiles on distributed systems (non z/OS)	57
3.3.1 Creating a deployment manager profile	59
3.3.2 Creating an application server profile	67
3.3.3 Creating a cell profile	78
3.3.4 Creating a custom profile	79
3.3.5 Federating a custom node to a cell	86
3.3.6 Creating a new application server on an existing node	88
3.3.7 Federating an application server profile to a cell	91
3.4 Creating profiles on z/OS systems	95
3.5 Managing profiles	123
3.5.1 Using the manageprofiles command	123
3.5.2 Creating a profile	124
3.5.3 Deleting profiles	126
3.6 Managing the processes	127
3.6.1 Starting a distributed server environment	128
3.6.2 Stopping the distributed server environment	129
3.6.3 Enabling process restart on failure	130
Chapter 4. Administration basics	137
4.1 Introducing the WebSphere administrative console	138
4.1.1 Starting the administrative console	138
4.1.2 Logging in to the administrative console	140
4.1.3 Changing the administrative console session timeout	142
4.1.4 The graphical interface	143
4.1.5 Finding an item in the console	147
4.1.6 Updating existing items	151
4.1.7 Adding new items	153
4.1.8 Removing items	154
4.1.9 Starting and stopping items	154
4.1.10 Using variables	156
4.1.11 Saving work	157
4.1.12 Getting help	158
4.2 Securing the administrative console	159
4.3 Working with the deployment manager	162
4.3.1 Deployment manager configuration settings	162
4.3.2 Starting and stopping the deployment manager	166

4.4 Working with application servers	170
4.4.1 Creating an application server	171
4.4.2 Viewing the status of an application server	175
4.4.3 Starting an application server	178
4.4.4 Stopping an application server	181
4.4.5 Viewing run time attributes of an application server	185
4.4.6 Customizing application servers	188
4.5 Working with nodes	201
4.5.1 Adding (federating) a node	201
4.5.2 Removing a node	209
4.5.3 Renaming a node	212
4.5.4 Node agent synchronization	213
4.5.5 Starting and stopping nodes	215
4.5.6 Node groups	219
4.6 Working with clusters	222
4.6.1 Creating clusters	222
4.6.2 Viewing cluster topology	226
4.6.3 Managing clusters	226
4.7 Working with virtual hosts	227
4.7.1 Creating a virtual host	228
4.8 Managing applications	229
4.8.1 Using the administrative console to manage applications	230
4.8.2 Installing an enterprise application	231
4.8.3 Uninstalling an enterprise application	233
4.8.4 Exporting an enterprise application	233
4.8.5 Starting an enterprise application	234
4.8.6 Stopping an enterprise application	234
4.8.7 Preventing an enterprise application from starting on a server	234
4.8.8 Viewing application details	235
4.8.9 Finding a URL for a servlet or JSP	238
4.9 Managing your configuration files	242
4.9.1 Backing up a profile	243
4.9.2 Restoring a profile	244
4.9.3 Exporting and importing profiles	246
Chapter 5. Administration with scripting	249
5.1 Overview of WebSphere scripting	250
5.2 Using wsadmin	250
5.2.1 Jacl versus Jython	250
5.2.2 Launching wsadmin	251
5.2.3 Configuring wsadmin	252
5.2.4 Command and script invocation	254
5.2.5 Overview of wsadmin objects	257

5.2.6 Management using wsadmin objects	259
5.3 Common operational tasks using wsadmin	278
5.3.1 General approach for operational tasks	278
5.3.2 Examples of common administrative tasks	279
5.3.3 Managing the deployment manager	279
5.3.4 Managing nodes	280
5.3.5 Managing application servers	281
5.3.6 Managing enterprise applications	283
5.3.7 Managing clusters	285
5.3.8 Generating the Web server plug-in configuration	286
5.3.9 Enabling tracing for WebSphere components	286
5.4 Common configuration tasks	288
5.4.1 General approach for configuration tasks	288
5.4.2 Specific examples of WebSphere configuration tasks	288
5.5 Help creating wsadmin scripts	300
5.6 Using Java for administration	301
Online resources	302
Chapter 6. Configuring WebSphere resources	303
6.1 WebSphere resources	304
6.2 JDBC resources	305
6.2.1 What are JDBC providers and data sources?	306
6.2.2 WebSphere support for data sources	307
6.2.3 Creating a data source	311
6.2.4 Creating a JDBC provider	311
6.2.5 Creating JDBC data source	317
6.3 JCA resources	329
6.3.1 WebSphere Application Server JCA support	331
6.3.2 Installing and configuring resource adapters	333
6.3.3 Configuring J2C connection factories	337
6.3.4 Using resource adapters from an application	340
6.4 JavaMail resources	342
6.4.1 JavaMail sessions	343
6.4.2 Configuring the mail provider	343
6.4.3 Configuring JavaMail sessions	347
6.4.4 Example code	350
6.5 URL providers	351
6.5.1 Configuring URL providers	351
6.5.2 Configuring URLs	353
6.5.3 URL provider sample	354
6.6 Resource environment providers	355
6.6.1 Resource environment references	356
6.6.2 Configuring the resource environment provider	357

6.7 Resource authentication	361
6.8 More information	363
Chapter 7. Managing Web servers.	365
7.1 Web server support overview	366
7.1.1 Request routing using the plug-in	366
7.1.2 Web server and plug-in management	367
7.2 Working with Web servers	371
7.2.1 Defining nodes and Web servers	371
7.2.2 Viewing the status of a Web server	376
7.2.3 Starting and stopping a Web server	376
7.2.4 IBM HTTP Server remote administration	378
7.2.5 Mapping modules to servers	383
7.3 Working with the plug-in configuration file	385
7.3.1 Regenerating the plug-in configuration file	386
7.3.2 Propagating the plug-in configuration file	392
7.3.3 Modifying the plug-in request routing options	393
Part 2. Messaging with WebSphere.	397
Chapter 8. Asynchronous messaging	399
8.1 Messaging concepts	400
8.1.1 Loose coupling	400
8.1.2 Messaging types	401
8.1.3 Destinations	401
8.1.4 Messaging models	402
8.1.5 Messaging patterns	404
8.2 Java Message Service	406
8.2.1 JMS API history	406
8.2.2 JMS providers	407
8.2.3 JMS domains	407
8.2.4 JMS administered objects	408
8.2.5 JMS and JNDI	409
8.2.6 JMS Connections	411
8.2.7 JMS sessions	412
8.2.8 JMS messages	413
8.2.9 JMS message producers	415
8.2.10 JMS message consumers	415
8.2.11 JMS exception handling	419
8.2.12 Application Server Facilities	421
8.2.13 JMS and J2EE	422
8.3 Messaging in the J2EE Connector Architecture	422
8.3.1 Message endpoints	425
8.3.2 MessageEndpointFactory	425

8.3.3	Resource adapters	425
8.3.4	JMS ActivationSpec JavaBean	428
8.3.5	Message endpoint deployment	430
8.3.6	Message endpoint activation	431
8.3.7	Message delivery	432
8.3.8	Administered objects	433
8.4	Message-driven beans	434
8.4.1	Message-driven bean types	434
8.4.2	Client view of a message-driven bean	435
8.4.3	Message-driven bean implementation	435
8.4.4	Message-driven bean life cycle	437
8.4.5	Message-driven beans and transactions	439
8.4.6	Message-driven bean activation configuration properties	443
8.4.7	Associating a message-driven bean with a destination	445
8.4.8	Message-driven bean best practices	448
8.5	Managing WebSphere JMS providers	451
8.5.1	Managing the default messaging JMS provider	451
8.5.2	Managing the WebSphere MQ JMS provider	455
8.5.3	Managing a generic JMS provider	457
8.6	Configuring WebSphere JMS administered objects	461
8.6.1	Common administration properties	462
8.6.2	Configuring the default messaging JMS provider	462
8.6.3	Configuring the WebSphere MQ JMS provider	491
8.6.4	Configuring listener ports	511
8.6.5	Configuring a generic JMS provider	515
8.7	Connecting to a service integration bus	520
8.7.1	JMS client run time environment	521
8.7.2	Controlling messaging engine selection	524
8.7.3	Load balancing bootstrapped clients	534
8.8	References and resources	536
Chapter 9	Default messaging provider	539
9.1	Concepts and architecture	540
9.1.1	Buses	540
9.1.2	Bus members	541
9.1.3	Messaging engines	541
9.1.4	Message stores	547
9.1.5	Destinations	549
9.1.6	Mediations	555
9.1.7	Foreign buses	555
9.2	Run time components	561
9.2.1	SIB service	561
9.2.2	Service integration bus transport chains	563

9.2.3	Message stores	568
9.2.4	Exception destinations	579
9.2.5	Service integration bus links	580
9.2.6	WebSphere MQ links	584
9.2.7	WebSphere MQ Servers	592
9.3	High availability and workload management	594
9.3.1	Cluster bus members for high availability	594
9.3.2	Cluster bus members for workload management	594
9.3.3	Partitioned queues	595
9.3.4	JMS clients connecting into a cluster of messaging engines	596
9.3.5	Preferred servers and core group policies	597
9.3.6	Best practices	600
9.4	Service integration bus topologies	601
9.4.1	One server in the cell is a member of one bus	601
9.4.2	Every server in the cell is a member of the same bus	602
9.4.3	A single cluster bus member and one messaging engine	602
9.4.4	A cluster bus member with multiple messaging engines	603
9.4.5	Mixture of cluster and server bus members	603
9.4.6	Multiple buses in a cell	604
9.5	Service integration bus and message-driven beans	605
9.5.1	Message-driven beans connecting to the bus	605
9.5.2	MDBs and clusters	607
9.6	Service integration bus security	608
9.7	Problem determination	610
9.8	Configuration and management	612
9.8.1	SIB service configuration	613
9.8.2	Creating a bus	616
9.8.3	Configuring bus properties	616
9.8.4	Enabling bus security	618
9.8.5	Adding a bus member	622
9.8.6	Creating a queue destination	637
9.8.7	Creating a topic space destination	639
9.8.8	Creating an alias destination	640
9.8.9	Adding messaging engines to a cluster	642
9.8.10	Setting up preferred servers	643
9.8.11	Setting up a foreign bus link to a service integration bus	650
9.8.12	Setting up a foreign bus link to an MQ queue manager	656
9.8.13	Creating a foreign destination	666
Part 3	Working with applications	669
Chapter 10	Session management	671
10.1	HTTP session management	672

10.2	Session manager configuration	672
10.2.1	Session management properties	672
10.2.2	Accessing session management properties	673
10.3	Session scope	674
10.4	Session identifiers	676
10.4.1	Choosing a session tracking mechanism	676
10.4.2	SSL ID tracking	678
10.4.3	Cookies	679
10.4.4	URL rewriting	682
10.5	Local sessions	683
10.6	General properties for session management	685
10.7	Session affinity	688
10.7.1	Session affinity and failover	690
10.8	Persistent session management	692
10.8.1	Enabling database persistence	694
10.8.2	Memory-to-memory replication	698
10.8.3	Session management tuning	708
10.8.4	Persistent sessions and non-serializable J2EE objects	715
10.8.5	Larger DB2 page sizes and database persistence	716
10.8.6	Single and multi-row schemas (database persistence).	717
10.8.7	Contents written to the persistent store using a database	719
10.9	Invalidating sessions	723
10.9.1	Session listeners	723
10.10	Session security	725
10.11	Session performance considerations	726
10.11.1	Session size	727
10.11.2	Reducing persistent store I/O	730
10.11.3	Multirow persistent sessions: Database persistence	731
10.11.4	Managing your session database connection pool	732
10.11.5	Session database tuning	733
10.12	Stateful session bean failover	734
10.12.1	Enabling stateful session bean failover	734
10.12.2	Stateful session bean failover considerations	737
Chapter 11.	WebSphere naming implementation	741
11.1	Features	742
11.2	WebSphere naming architecture	743
11.2.1	Components	743
11.2.2	JNDI support	744
11.2.3	JNDI bindings	745
11.2.4	Federated name space	746
11.2.5	Local name space structure	749
11.3	Interoperable Naming Service (INS)	757

11.3.1 Bootstrap ports	757
11.3.2 CORBA URLs	757
11.4 Distributed CosNaming	759
11.5 Configured bindings	760
11.5.1 Types of objects	761
11.5.2 Types of binding references	762
11.6 Initial contexts	763
11.6.1 Setting initial root context	766
11.7 Federation of name spaces	769
11.8 Foreign cell bindings	770
11.9 Interoperability	771
11.9.1 WebSphere V4.0 EJB clients	772
11.9.2 WebSphere V4.0 server	773
11.9.3 EJB clients hosted by non-WebSphere environment	773
11.10 Examples	774
11.10.1 Single server	775
11.10.2 Two single servers on the same box	776
11.10.3 Network Deployment application servers on the same box	778
11.10.4 WebSphere Application Server V4 client	780
11.11 Naming tools	782
11.11.1 dumpNameSpace	782
11.12 Configuration	784
11.12.1 Name space bindings	785
11.12.2 Foreign cell bindings	788
11.12.3 CORBA naming service users and groups	789
Chapter 12. Understanding class loaders	795
12.1 A brief introduction to Java class loaders	796
12.2 WebSphere class loaders overview	800
12.2.1 WebSphere extensions class loader	801
12.2.2 Application and Web module class loaders	802
12.2.3 Handling JNI code	803
12.3 Configuring WebSphere for class loaders	804
12.3.1 Class loader policies	804
12.3.2 Class loading/delegation mode	807
12.3.3 Shared libraries	809
12.4 Class loader viewer	810
12.5 Learning class loaders by example	811
12.5.1 Step 1: Simple Web module packaging	812
12.5.2 Step 2: Adding an EJB module and Utility jar	817
12.5.3 Step 3: Changing the WAR class loader delegation mode	818
12.5.4 Step 4: Sharing utility JARs using shared libraries	820
12.6 Additional class loader diagnostics	827

Chapter 13. Packaging applications	829
13.1 Plants by WebSphere sample application	830
13.1.1 Plants by WebSphere resources used	831
13.2 Packaging using the Application Server Toolkit	832
13.2.1 Import source code	832
13.2.2 Working with deployment descriptors	838
13.3 Setting application bindings	842
13.3.1 Defining EJB JNDI names	842
13.3.2 Binding EJB and resource references	844
13.3.3 Defining data sources for entity beans	846
13.3.4 Setting the context root for Web modules	853
13.4 IBM EJB extensions: EJB caching options	854
13.4.1 EJB container caching option for entity beans	854
13.4.2 EJB container caching option for stateful session beans	858
13.4.3 Stateful EJB timeout option	859
13.5 IBM EJB extensions: EJB access intents	860
13.5.1 Transaction isolation levels overview	860
13.5.2 Concurrency control	862
13.5.3 Using EJB 2.x access intents	863
13.5.4 Using read-ahead hints	868
13.5.5 Tracing access intents behavior	871
13.6 IBM EJB extensions: inheritance relationships	871
13.7 IBM Web module extensions	872
13.7.1 File serving servlet	872
13.7.2 Web application auto reload	873
13.7.3 Serve servlets by class name	873
13.7.4 Default error page	874
13.7.5 Directory browsing	874
13.7.6 JSP attributes	874
13.7.7 Automatic HTTP request and response encoding	875
13.8 IBM EAR extensions: Sharing session context	876
13.9 Exporting the PlantsByWebSphere EAR file	877
13.10 WebSphere Enhanced EAR	877
13.10.1 Configuring a WebSphere Enhanced EAR	879
13.11 Packaging recommendations	891
Chapter 14. Deploying applications	893
14.1 Preparing the environment	894
14.1.1 Creating the Plants by WebSphere DB2 database	894
14.1.2 Creating an environment variable	895
14.1.3 Creating the Plants by WebSphere application server	896
14.1.4 Defining the Plants by WebSphere virtual host	901
14.1.5 Creating the virtual host for IBM HTTP Server and Apache	902

14.1.6 Creating a DB2 JDBC provider and data source	904
14.2 Generating deployment code	911
14.2.1 Using EJBDeploy command-line tool	911
14.3 Deploying the application	913
14.3.1 Using a bindings file	919
14.4 Deploying application clients	920
14.4.1 Defining application client bindings	924
14.4.2 Launching the J2EE client	925
14.5 Updating applications	928
14.5.1 Replacing an entire application EAR file	928
14.5.2 Replacing or adding an application module	929
14.5.3 Replacing or adding single files in an application or module	930
14.5.4 Removing application content	931
14.5.5 Performing multiple updates to an application or module	931
14.5.6 Rolling out application updates to a cluster	934
14.5.7 Hot deployment and dynamic reloading	938
Related publications	941
IBM Redbooks	941
Other publications	942
Online resources	942
How to get IBM Redbooks	944
Help from IBM	944
Index	945

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX 5L™	ibm.com®	Redbooks™
AIX®	IBM®	Redbooks (logo)  ™
alphaWorks®	IMS™	RACF®
Cloudscape™	Informix®	SupportPac™
CICS®	iSeries™	System z™
Domino®	Lotus®	Tivoli®
DB2 Connect™	MVS™	WebSphere®
DB2 Universal Database™	MVS/ESA™	Workplace™
DB2®	OS/400®	z/OS®
e-business on demand®	Power PC®	zSeries®
i5/OS®	Rational®	

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

iPlanet, Enterprise JavaBeans, EJB, Java, Java Naming and Directory Interface, Javadoc, JavaBeans, JavaMail, JavaServer, JavaServer Pages, JDBC, JDK, JMX, JRE, JSP, JVM, J2EE, J2SE, Solaris, Streamline, Sun, Sun Java, Sun Microsystems, Sun ONE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Active Directory, ActiveX, Microsoft, Visual Basic, Windows NT, Windows Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbook provides system administrators, developers, and architects with the knowledge to configure a WebSphere Application Server V6.1 run time environment, to package and deploy Web applications, and to perform ongoing management of the WebSphere® environment.

One in a series of handbooks, the entire series is designed to give you in-depth information about the entire range of WebSphere Application Server products. In this IBM Redbook, we provide a detailed exploration of the WebSphere Application Server V6.1 run time environments and administration process.

The IBM Redbook includes configuration and administration information for WebSphere Application Server V6.1 and WebSphere Application Server Network Deployment V6.1 on distributed platforms (excluding iSeries™) and WebSphere Application Server for z/OS® V6.1.

The following are considered companion pieces to this IBM Redbook:

- ▶ *WebSphere Application Server V6.1: Technical Overview*, REDP-4191 at:
<http://www.redbooks.ibm.com/redpieces/abstracts/redp4191.html>
- ▶ *WebSphere Application Server V6.1: Planning and Design*, SG24-7305 at:
<http://www.redbooks.ibm.com/redpieces/abstracts/sg247305.html>

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Carla Sadtler is a certified IT Specialist at the ITSO, Raleigh Center. She writes extensively about the WebSphere and Patterns for e-business areas. Before joining the ITSO in 1985, Carla worked in the Raleigh branch office as a Program Support Representative. She holds a degree in mathematics from the University of North Carolina at Greensboro.

Fabio Albertoni is a Senior IT Specialist working from Integrated Technology Delivery SSO, on Hortolandia, Brazil. He has nine years of experience work in IT Industry and Banks, he has spent last five years developing and implementing integrated solutions using WebSphere Application Server and MQ-Series. He hold a degree in Data Process from FATEC University of Ourinhos and a Master degree on Computer Engineer from Instituto de Pesquisas Tecnologicas of Sao Paulo, Brazil.

Bernardo Fagalde is an IT Architect at IBM Uruguay, working for IBM since 2000. During his time at IBM, he has had many positions, including database administrator, system administrator, developer, designer, application server administrator, and as a technical lead for e-business projects. He has been working with WebSphere Application Server since V3.5 and mostly designs e-business solutions focused on using the WebSphere product family. He is currently the lead IT Architect on a large J2EE™ project. Bernardo holds a Computing Engineer degree from the Uruguayan main University (Universidad de la República Oriental del Uruguay).

Thiago Kleinubing is an IT Specialist in Brazil and has over nine years of experience in the IT field. He has been working at IBM for the last six years and is currently a Team Leader for the IBM Global Business Services Organization - Total Workplace™ Experience Center of Excellence. His areas of expertise include the architecture, design, and development of J2EE applications. He is also an expert in IBM WebSphere Application Server, performance tuning, and problem determination. Thiago holds a degree in computer science and is certified in Rational® Application Developer and WebSphere Studio V5.

Henrik Sjostrand is a Senior IT Specialist and has worked for IBM Sweden for 12 years. He is currently working as a technical consultant for the Nordic IBM Software Services for WebSphere team. For the last six years, he has focused on J2EE application development and WebSphere Application Server architecture, deployment, performance tuning, and troubleshooting. He is certified in WebSphere Application Server V4, V5, and V6, WebSphere Studio V5, and Rational Application Developer V6. Henrik holds a Master of Science in Electrical Engineering from Chalmers University of Technology in Gothenburg, Sweden, where he lives.

Ken Worland is a senior IT Specialist based in Melbourne, Australia. He specializes in Web services and messaging solutions and has over 15 years of experience in the IT field. His areas of expertise include WebSphere Application Server, WebSphere MQ, DB2®, Oracle, and much more, having worked as a UNIX® system administrator and database administrator on occasion. Ken holds a Bachelors degree in Computer Science from LaTrobe University in Melbourne.

Special thanks to the authors of the previous book, *WebSphere Application Server V6 System Management & Configuration Handbook*, SG24-6451:

Lars Bek Laursen was an Advisory IT Specialist at the Integrated Technology Services division of IBM Global Services in Lyngby, Denmark while working on this project. He has eight years of Java™ experience, from developing Java-based systems management solutions to designing and implementing enterprise application server environments. For the last five years, Lars has worked extensively as a WebSphere Application Server consultant, advising on problem solving, tuning, and implementation of fail-safe run time environments. Lars holds a Master of Science in Engineering degree from the Technical University of Denmark. Lars has since left IBM.

Martin Phillips is a tester for the WebSphere Messaging and Transaction Technology team in the Hursley Laboratories in the UK. He has worked for IBM UK for five years as a tester for WebSphere Application Server. His areas of expertise include the service integration bus, about which he writes extensively in this book. Martin holds a Master of Science in Information Technology specializing in Software and Systems from the University of Glasgow.

Martin Smithson is a Senior IT Specialist working for IBM Software Group in Hursley, England. He has nine years of experience working in the IT Industry and has spent the last four years working as a technical consultant for the EMEA IBM Software Services for WebSphere team. He is certified in WebSphere Application Server V3.5, V4 and V5 and WebSphere Studio Application Developer V4.0.3 and V5. His areas of expertise include the architecture, design, and development of J2EE applications. He is also an expert on IBM WebSphere Application Server. He has written extensively on asynchronous messaging and the service integration bus. He holds a degree in Software Engineering from City University in London, UK.

Kwan-Ming Wan is a Consulting IT Specialist working for the IBM Software Group in London, England. He has over fifteen years of experience in the IT industry and has been working as a consulting professional throughout his career. For the past five years, he has been working as a WebSphere consultant with a focus on performance tuning, problem determination, and architecture design. He holds a Master of Science degree in Information Technology from the University of Nottingham, England.

Thanks to the following people for their contributions to this project:

Margaret Ticknor
International Technical Support Organization, Raleigh Center

Rich Conway
International Technical Support Organization, Poughkeepsie Center

Mollie Tucker
IBM Intern from North Carolina State University

Daniel Tishman
IBM Intern from Penn State University

Sam Cleveland
WebSphere Application Server Samples Development

The following members of the WebSphere Messaging and Transaction
Technologies Team, IBM Hursley:

Malcolm Ayres
David Currie
Sarah Hemmings
Graham Hopkins
Geraint Jones
Adrian Preston
Anne Redwood
Matthew Vaughton
Graham Wallis



Figure 1 Authors: (from left to right) Fabio Albertoni, Carla Sadtler, Thiago Kleinubing, Bernardo Fagalde, Ken Worland, Henrik Sjostrand

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and client satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an e-mail to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400



Part 1

The basics

This part introduces you to WebSphere Application Server V6.1. It includes information about the run time architecture, administration tools, and the basics of configuring and managing the run time environment.

This part includes the following:

- ▶ Chapter 1, “WebSphere Application Server” on page 3
- ▶ Chapter 2, “System management: A technical overview” on page 15
- ▶ Chapter 3, “Getting started with profiles” on page 47
- ▶ Chapter 4, “Administration basics” on page 137
- ▶ Chapter 5, “Administration with scripting” on page 249
- ▶ Chapter 6, “Configuring WebSphere resources” on page 303
- ▶ Chapter 7, “Managing Web servers” on page 365

WebSphere Application Server

IBM WebSphere is the leading software platform for e-business on demand®. Providing comprehensive e-business leadership, WebSphere is evolving to meet the demands of companies faced with challenging business requirements, such as the need for increasing operational efficiencies, strengthening client loyalty, and integrating disparate systems. WebSphere provides answers in today's challenging business environments.

IBM WebSphere is architected to enable you to build business-critical applications for the Web. WebSphere includes a wide range of products that help you develop and serve Web applications. They are designed to make it easier for clients to build, deploy, and manage dynamic Web sites more productively.

In this chapter, we take a look at the new WebSphere Application Server V6.1 for distributed platforms and WebSphere Application Server for z/OS.

1.1 Product overview

WebSphere is the IBM brand of software products designed to work together to help deliver dynamic e-business quickly. It provides solutions for connecting people, systems, and applications with internal and external resources. WebSphere is based on infrastructure software, or middleware, designed for dynamic e-business. It delivers a proven, secure, and reliable software portfolio that can provide an excellent return on investment.

The technology that powers WebSphere products is Java. Over the years, many software vendors have collaborated on a set of server-side application programming technologies that help build Web accessible, distributed, and platform-neutral applications. These technologies are collectively branded as the Java 2 Platform, Enterprise Edition (J2EE) platform. This contrasts with the Java 2 Standard Edition (J2SE™) platform, with which most clients are familiar. J2SE supports the development of client-side applications with rich graphical user interfaces (GUIs). The J2EE platform is built on top of the J2SE platform. J2EE consists of application technologies for defining business logic and accessing enterprise resources, such as databases, Enterprise Resource Planning (ERP) systems, messaging systems, e-mail servers, and so forth.

The potential value of J2EE to clients is tremendous. Among the benefits of J2EE are:

- ▶ An architecture-driven approach to application development helps reduce maintenance costs and allows for construction of an information technology (IT) infrastructure that can grow to accommodate new services.
- ▶ Application development is focused on unique business requirements and rules, such as security and transaction support. This improves productivity and shortens development cycles.
- ▶ Industry standard technologies allow clients to choose among platforms, development tools, and middleware to power their applications.
- ▶ Embedded support for Internet and Web technologies allows for a new breed of applications that can bring services and content to a wider range of customers, suppliers, and others, without creating the need for proprietary integration.

Another exciting opportunity for IT is Web services. Web services allow for the definition of functions or services within an enterprise that can be accessed using industry standard protocols that most businesses already use today, such as HTTP and XML. This allows for easy integration of both intra- and inter-business applications that can lead to increased productivity, expense reduction, and quicker time to market.

1.2 WebSphere Application Server

WebSphere Application Server provides the environment to run your Web-enabled e-business applications. An application server functions as *Web middleware* or a middle tier in a three-tier e-business environment. The first tier is the HTTP server that handles requests from the browser client. The third tier is the business database (for example, DB2 UDB for iSeries) and the business logic (for example, traditional business applications, such as order processing). The middle tier is WebSphere Application Server, which provides a framework for a consistent and architected link between the HTTP requests and the business data and logic.

WebSphere Application Server is available on a wide range of platforms and in multiple packages to meet specific business needs. It also serves as the base for other WebSphere products, such as WebSphere Enterprise Service Bus and WebSphere Process Server, by providing the application server that is required to run these specialized applications.

Figure 1-1 illustrates a product overview of WebSphere Application Server.

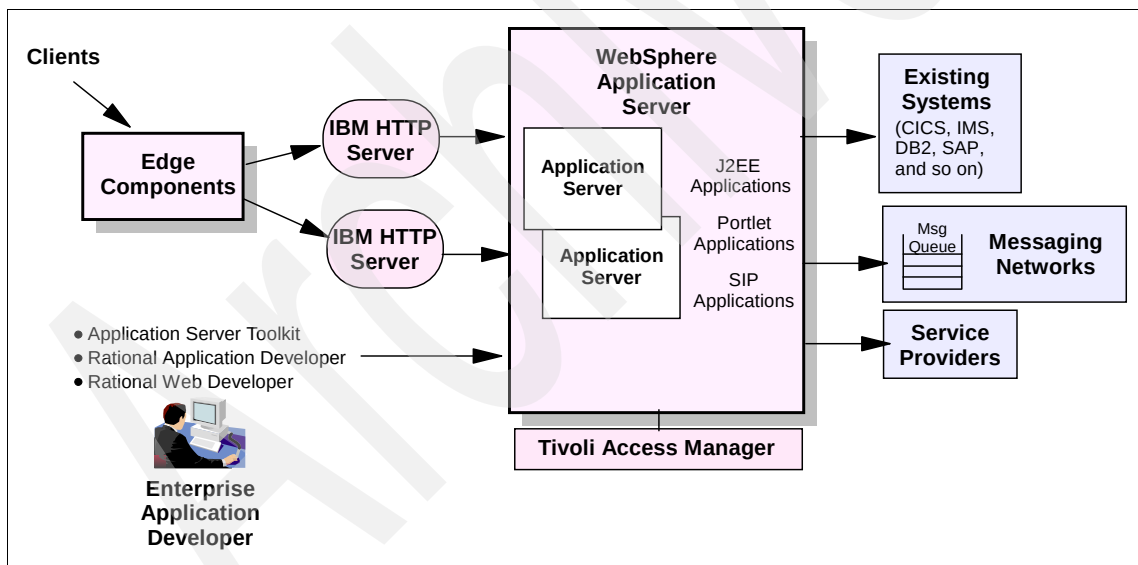


Figure 1-1 WebSphere Application Server product overview

The application server is the key component of WebSphere Application Server, providing the run time environment for applications that conform to the J2EE 1.2, 1.3, and 1.4 specifications. Clients access these applications through standard interfaces and APIs. The applications, in turn, have access to a wide variety of external sources, such as existing systems, databases, Web services, and

messaging resources that can be used to process the client requests. V6.1 extends the application server to allow it to run JSR 168 compliant portlets and Session Initiation Protocol (SIP) applications written to the JSR 116 specification.

With the Base and Express packages, you are limited to single application server environments. The Network Deployment package allows you to extend this environment to include multiple application servers that are administered from a single point of control and can be clustered to provide scalability and high availability environments.

WebSphere Application Server supports asynchronous messaging through the use of a JMS provider and its related messaging system. WebSphere Application Server includes a fully integrated JMS 1.1 provider called the default messaging provider. This messaging provider complements and extends WebSphere MQ and application server. It is suitable for messaging among application servers and for providing messaging capability between WebSphere Application Server and an existing WebSphere MQ backbone.

WebSphere Application Server provides authentication and authorization capabilities to secure administrative functions and applications. Your choice of user registries include the operating system user registry, an LDAP registry (for example, Tivoli® Directory Server), custom registries, file-based registries, or federated repositories. In addition to the default authentication and authorization capabilities, you have the option of using an external Java Authorization Contract for Containers (JACC) compliant authorization provider for application security. The IBM Tivoli Access Manager client embedded in WebSphere Application Server is JACC-compliant and can be used to secure your WebSphere Application Server-managed resources. This client technology is designed to be used with the Tivoli Access Manager Server (shipped with Network Deployment).

WebSphere Application Server works with a Web server (such as the IBM HTTP Server) to route requests from browsers to the applications that run in WebSphere Application Server. Web server plug-ins are provided for installation with supported Web browsers. The plug-ins direct requests to the appropriate application server and perform workload balancing among servers in a cluster.

WebSphere Application Server Network Deployment includes the Caching Proxy and Load Balancer components of Edge Component for use in highly available, high volume environments. Using these components can reduce Web server congestion, increase content availability, and improve Web server performance.

1.3 Packaging

Because varying e-business application scenarios require different levels of application server capabilities, WebSphere Application Server is available in multiple packaging options. Although they share a common foundation, each provides unique benefits to meet the needs of applications and the infrastructure that supports them. At least one WebSphere Application Server product fulfills the requirements of any particular project and its supporting infrastructure. As your business grows, the WebSphere Application Server family provides a migration path to more complex configurations.

WebSphere Application Server - Express V6.0

The Express package is geared to those who need to get started quickly with e-business. It is specifically targeted at medium-sized businesses or departments of a large corporation, and is focused on providing ease of use and ease of application development. It contains full J2EE 1.4 support but is limited to a single-server environment.

WebSphere Application Server - Express is unique from the other packages in that it is bundled with an application development tool. Although there are WebSphere Studio and Rational Developer products designed to support each WebSphere Application Server package, normally they are ordered independent of the server. WebSphere Application Server - Express includes the Rational Web Developer application development tool. It provides a development environment geared toward Web developers and includes support for most J2EE 1.4 features with the exception of Enterprise JavaBeans™ (EJB™) and J2EE Connector Architecture (JCA) development environments. However, keep in mind that WebSphere Application Server - Express V6 does contain full support for EJB and JCA, so you can deploy applications that use these technologies.

WebSphere Application Server V6.1

The WebSphere Application Server package is the next level of server infrastructure in the WebSphere Application Server family. Though the WebSphere Application Server is functionally equivalent to that shipped with Express, this package differs slightly in packaging and licensing.

This package includes two tools for application development and assembly:

- The Application Server Toolkit, which has been expanded in V6.1 to include a full set of development tools. The toolkit is suitable for J2EE 1.4 application development as well as the assembly and deployment of J2EE applications. It also supports Java 5 development.

In addition, the toolkit provides tools for the development, assembly, and deployment of JSR 116 SIP and JSR 168 portlet applications.

- ▶ This package also includes a trial version of Rational Application Developer, which supports the development, assembly, and deployment of J2EE 1.4 applications.

To avoid confusion with the Express package in this IBM Redbook, we refer to this as the Base package.

WebSphere Application Server Network Deployment V6

WebSphere Application Server Network Deployment is an even higher level of server infrastructure in the WebSphere Application Server family. It extends the WebSphere Application Server base package to include clustering capabilities, Edge components, and high availability for distributed configurations. These features become more important at larger enterprises, where applications tend to service a larger client base, and more elaborate performance and availability requirements are in place.

Application servers in a cluster can reside on the same or multiple machines. A Web server plug-in installed in the Web server can distribute work among clustered application servers. In turn, Web containers running servlets and Java ServerPages (JSPs) can distribute requests for EJBs among EJB containers in a cluster.

The addition of Edge components provides high performance and high availability features. For example:

- ▶ The Caching Proxy intercepts data requests from a client, retrieves the requested information from the application servers, and delivers that content back to the client. It stores cachable content in a local cache before delivering it to the client. Subsequent requests for the same content are served from the local cache, which is much faster and reduces the network and application server load.
- ▶ The Load Balancer provides horizontal scalability by dispatching HTTP requests among several, identically configured Web server or application server nodes.

WebSphere Application Server V6.1 for z/OS

IBM WebSphere Application Server for z/OS is a full-function version of the Network Deployment product. WebSphere Application Server for z/OS can support e-business on any scale.

Packaging summary

Table 1-1 shows the features included with each WebSphere Application Server packaging option.

Table 1-1 WebSphere Application Server packaging

Features included	Express V6.0 ¹	Base V6.1	Network Deployment V6.1	V6.1 for z/OS
WebSphere Application Server	Yes	Yes	Yes	Yes
Deployment manager	No	No	Yes	Yes
Web server plug-ins	Yes	Yes	Yes	Yes
IBM HTTP Server	Yes	Yes	Yes	Yes
Application Client (not available on Linux® for zSeries®)	Yes	Yes	Yes	Yes
Application Server Toolkit	Yes	Yes	Yes	Yes
DataDirect Technologies JDBC™ Drivers for WebSphere Application Server	Yes	Yes	Yes	Yes (for Windows® only)
Rational Development tools	Rational Web Developer (single use license)	Rational Application Developer Trial	Rational Application Developer Trial	Rational Application Developer Trial (non-z/OS platforms)
Database	IBM DB2 Universal Database™ Express V8.2	IBM DB2 Universal Database Express V8.2 (development use only)	IBM DB2 UDB Enterprise Server Edition V8.2 for WebSphere Application Server Network Deployment	No
Production ready applications	IBM Business Solutions	No	No	No
Tivoli Directory Server for WebSphere Application Server (LDAP server)	No	No	Yes	No

Features included	Express V6.0 ¹	Base V6.1	Network Deployment V6.1	V6.1 for z/OS
Tivoli Access Manager Servers for WebSphere Application Server	No	No	Yes	Yes (non-z/OS platforms)
Edge Components	No	No	Yes	Yes (non-z/OS platforms)
1. Express is limited to a maximum of two CPUs.				

Note: Not all features are available on all platforms. See the System Requirements Web page for each WebSphere Application Server package for more information.

1.4 Supported platforms and software

The following tables illustrate the platforms, software, and versions that WebSphere Application Server V6 supports at the time of the writing of this document. For the most up-to-date operating system levels and requirements, refer to the WebSphere Application Server system requirements at:

<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>

1.4.1 Operating systems

Table 1-2 shows the supported operating systems and versions for WebSphere Application Server V6.1.

Table 1-2 Supported operating systems and versions

Operating Systems	Versions
Windows	▶ Windows 2000 Advanced Server with SP4 ▶ Windows 2000 Server with SP4 ▶ Windows 2000 Professional Server with SP4 ▶ Microsoft® Windows Server® 2003 (Datacenter with SP1) ▶ Microsoft Windows Server 2003 (Enterprise with SP1) ▶ Microsoft Windows Server 2003 (Standard with SP1) ▶ Microsoft Windows XP Professional with SP2 ▶ Microsoft Windows Server 2003 x64 Editions
AIX®	▶ AIX 5L™ V5.2 Maintenance Level 5200-07 ▶ AIX 5L V5.3 with Service Pack 5300-04-01

Operating Systems	Versions
Sun™ Solaris™	▶ Solaris 9 with the latest patch Cluster ▶ Solaris 10 with the latest patch Cluster
HP-UX	▶ HP-UX 11iv2 (11.23) with the latest Quality Pack
Linux (Intel®)	▶ Red Hat Linux Enterprise AS, ES, WS V3 with Update 5 or 6 ▶ Red Hat Linux Enterprise AS, ES, WS V4 with Update 2 ▶ SUSE Linux Enterprise Server, V9 with SP2 or 3
Linux (Power PC®)	▶ Red Hat Enterprise Linux AS V3 with Update 5 or 6 ▶ Red Hat Enterprise Linux AS V4 with Update 2 ▶ SUSE Linux Enterprise Server V9 with SP2 or 3
Linux on IBM System z™ (Supported for WebSphere Application Server Network Deployment only)	▶ Red Hat Enterprise Linux AS V3 with Update 5 or 6 ▶ Red Hat Enterprise Linux AS V4 with Update 2 ▶ SUSE Linux Enterprise Server V9 with SP2 or 3
i5/OS® and OS/400®	▶ i5/OS and OS/400, V5R3 ▶ i5/OS V5R4
z/OS (Supported for WebSphere Application Server Network Deployment only)	▶ z/OS 1.6 or later ▶ z/OS.e 1.6 or later

1.4.2 Web servers

The following Web servers are supported by WebSphere Application Server V6.1 on all available platforms:

- ▶ Apache HTTP Server 2.0.54
- ▶ IBM HTTP Server for WebSphere Application Server V6.0.2
- ▶ IBM HTTP Server for WebSphere Application Server V6.1
- ▶ Internet Information Services 5.0
- ▶ Internet Information Services 6.0
- ▶ Lotus® Domino® Enterprise Server 6.5.4 or 7.0
- ▶ Sun Java™ System Web Server 6.0 SP9
- ▶ Sun Java System Web Server 6.1 SP3

1.4.3 Database servers

Table 1-3 shows the database servers that WebSphere Application Server V6.1 supports.

Table 1-3 Supported database servers and versions

Databases	Versions
IBM DB2	DB2 for iSeries 5.2, 5.3, or 5.4 DB2 for z/OS V7 or V8 DB2 Enterprise Server Edition 8.2 FP4 DB2 Express 8.2 FP4 DB2 Workgroup Server Edition 8.2 FP4
Cloudscape™	Cloudscape 10.1
Oracle	Oracle 9i Standard/Enterprise Release 2 - 9.2.0.7 Oracle 10g Standard/Enterprise Release 1 - 10.1.0.4 Oracle 10g Standard/Enterprise Release 2 - 10.2.0.1 or 10.2.0.2
Sybase	Sybase Adaptive Server Enterprise 12.5.2 or 15.0
Microsoft SQL Server	Microsoft SQL Server Enterprise 2000 SP4 Microsoft SQL Server Enterprise 2005
Informix®	Informix Dynamic Server 9.4C7W1 or 10.00C4
IMS™	IMS V8 or V9
WebSphere Information Integrator	WebSphere Information Integrator 8.2 FP4

1.4.4 Directory servers

Table 1-4 shows the LDAP servers that WebSphere Application Server V6.1 supports.

Table 1-4 Supported Directory servers and versions

Directory Server	Versions
IBM Tivoli Directory Server	5.2 and 6.0
z/OS Security Server	1.6 and 1.7
z/OS.e Security Server	1.6 and 1.7
Lotus Domino Enterprise Server	6.5.4 and 7.0
Sun ONE™ Directory Server	5.1 SP4 and 5.2

Directory Server	Versions
Windows Active Directory®	2003 and 2000
Novell eDirectory	8.7.3 and 8.8

Packaging applications

In this chapter, we show you how to perform some common tasks involved in packaging a J2EE application. For this purpose, we will use the Plants by WebSphere sample application that ships with WebSphere Application Server and the Application Server Toolkit.

This chapter includes the following topics:

- ▶ Plants by WebSphere sample application
- ▶ Packaging using the Application Server Toolkit
- ▶ Setting application bindings
- ▶ IBM EJB extensions: EJB caching options
- ▶ IBM EJB extensions: EJB access intents
- ▶ IBM EJB extensions: inheritance relationships
- ▶ IBM Web module extensions
- ▶ IBM EAR extensions: Sharing session context
- ▶ Exporting the PlantsByWebSphere EAR file
- ▶ WebSphere Enhanced EAR
- ▶ Packaging recommendations

13.1 Plants by WebSphere sample application

Plants by WebSphere is an Internet storefront that specializes in the sale of plants and gardening tools. This Sample application uses many of the Java 2 Platform, Enterprise Edition (J2EE) and WebSphere Application Server functions, including enterprise beans, servlets, and JavaServer Pages (JSP) technology.

Using the Plants by WebSphere storefront, customers can open accounts, browse for items to purchase, view product details, and place orders. The Plants by WebSphere application uses container-managed persistence (CMP), container-managed relationships (CMR), stateless session beans, a stateful session bean, JSP pages, and servlets.

We will not go into details on how Plants by WebSphere application works. For more detailed information, refer to the WebSphere Information Center. If you have installed the sample applications on your system, you can also find information at:

<http://localhost:9080/WSsamples>

When Plants by WebSphere is installed and configured properly, it can be invoked at:

<http://localhost:9080/PlantsByWebSphere>

The Plants by WebSphere EAR file consists of the modules show in Figure 13-1.

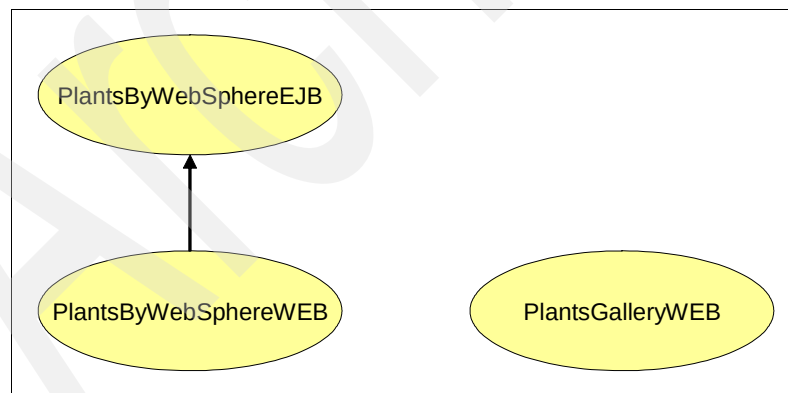


Figure 13-1 Plants By WebSphere modules

The Plants by WebSphere EAR file has one EJB module, and two Web modules. The PlantsByWebSphereWEB is the Web module where the servlets, JSPs, and

images are kept. The PlantsGalleryWEB module contains documentation for the sample, but no executables or Java code.

13.1.1 Plants by WebSphere resources used

To run successfully, Plants by WebSphere requires the following resources to be configured:

- ▶ JAAS authentication alias
 - Scope: Cell level
 - Name: <cellname>/samples
 - User ID: samples
 - Password: samples
- ▶ JDBC provider
 - Scope: Server level
 - Name: Samples Derby JDBC Provider (XA)
 - Implementation class: org.apache.derby.jdbc.EmbeddedXADataSource
- ▶ Data source
 - Scope: Server level
 - JDBC provider: Samples Derby JDBC Provider (XA)
 - Name: PLANTSDB
 - JNDI name: jdbc/PlantsByWebSphereDataSource
 - Database name:
\${APP_INSTALL_ROOT}/\${CELL}/PlantsByWebSphere.ear/Database/PLANTSDB
- ▶ Mail session:
 - Scope: Server level
 - Name: PlantsByWebSphere Mail Session
 - JNDI name: mail/PlantsByWebSphere
 - Type: javax.mail.Session

These are the resources configured when you install the samples during installation of WebSphere. Because we will change the Plants by WebSphere application to use an IBM DB2 database instead of the default Derby (Cloudscape) database, our resources will look a little bit different.

13.2 Packaging using the Application Server Toolkit

New in V6.1: The Application Server Toolkit shipped with V6.1 is now a full-blown integrated development environment (IDE). It can be used to build, test, and deploy J2EE applications on a WebSphere Application Server V6.1 environment (but not on any previous release). It has support for all J2EE artifacts supported by WebSphere Application Server V6.1, such as servlets, JSPs, EJBs, XML, Web services, and so on, and includes support for developing Java 5.0 applications.

To illustrate packaging techniques, we will import the source code for the Plants by WebSphere into the Application Server Toolkit, build it, and go through the various aspects of packaging an application for deployment. The application can then be exported as an EAR file for deployment.

The samples must be selected for installation during the WebSphere install. If you have installed the samples, the source code can be found in `<was_home>\samples\src\PlantsByWebSphere`.

13.2.1 Import source code

To work with the Plants by WebSphere sample application in the Application Server Toolkit, we first need to create an Enterprise Application project and then import the source code.

1. Select **Start** → **Programs** → **IBM WebSphere** → **Application Server Toolkit V6.1** → **Application Server Toolkit**.
2. When asked for a default location for a workspace, browse to a suitable directory and then click **OK**. Do not check the **Use this as the default and do not ask again** check box.
3. When the toolkit has launched, close the Welcome page by clicking the X in the Welcome tab, as in Figure 13-2.

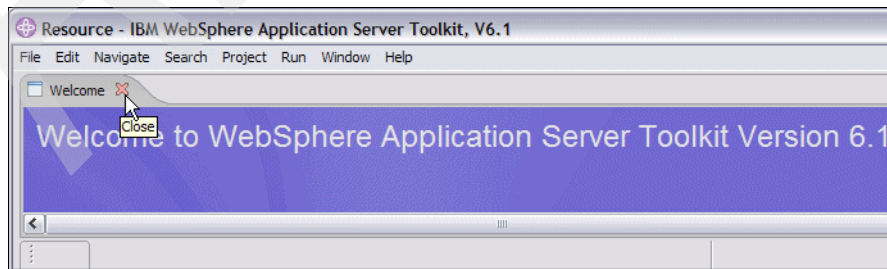


Figure 13-2 Welcome page

4. Select **File** → **New** → **Other**.
5. Expand **J2EE**, select **Enterprise Application Project**, and then click **Next**.
6. In the Project name field, enter PlantsByWebSphere, and then click **Next**.
7. On the Select Project Facets page, keep the defaults (EAR, WebSphere Application Co-existence, and WebSphere Application Extended), and click **Next**.
8. On the J2EE Modules to Add to the EAR page, click **New Module**.
9. On the New J2EE Module page, deselect the **Application client module** and **Connector module**. Keep the PlantsByWebSphereEJB and PlantsByWebSphereWeb projects as is. Then click **Finish**.
10. Back on the J2EE Modules to Add to the EAR, click **Finish**.
11. When the projects have been created, click **Yes** on the dialog box asking you to switch to the J2EE perspective.

You should now see the J2EE perspective with the Project Explorer view, as shown in Figure 13-3 on page 834.

Note that we did not create a project for the PlantsGalleryWeb module, because that only holds documentation and we do not need that for our example.

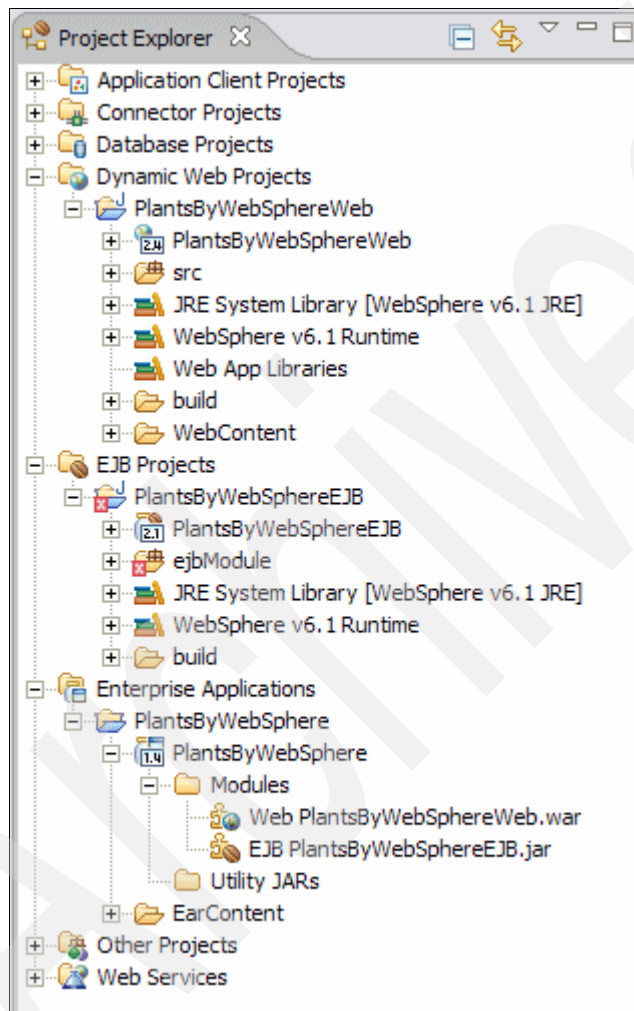


Figure 13-3 Project Explorer view with PlantsByWebSphere projects created

When the necessary projects are created, it is time to import the source code.

The defaults for Web modules is to keep source code in the src folder and the compiled Java classes in the build folder. For EJB projects, the source code is kept in the ejbModule folder and the compiled classes in the build folder.

We will start with the EJB source code first:

1. Select **File** → **Import**.
2. Select **File system** from the list, and then click **Next**.
3. For the From directory field, click **Browse** and select the <was_home>\samples\src\PlantsByWebSphere\PlantsByWebSphereEJB\ejbModule directory, and then click **OK**.
4. Select the **ejbModule** in the left pane.
5. For the Into folder field, click **Browse** and select the **PlantsByWebSphereEJB/ejbModule** folder, and click **OK**.
6. Keep the other options as is (Create selected folders only is selected only) and then click **Finish**. See Figure 13-4.

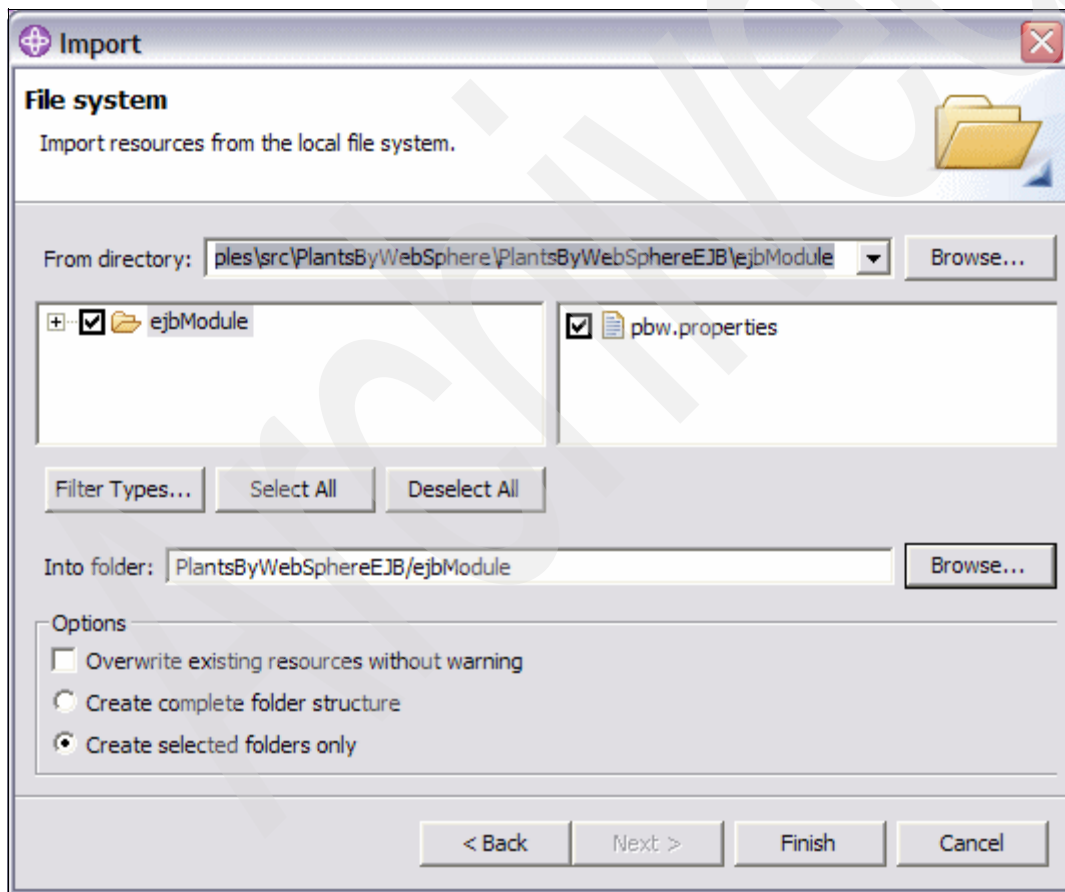


Figure 13-4 Importing EJB module source code

7. When the question about overwriting existing resources appear, click **Yes To All**.
8. Repeat the process to import the Web module source code from <was_home>\samples\src\PlantsByWebSphere\PlantsByWebSphereWEB\JavaSource into the PlantsByWebSphereWeb/src folder, and the Web content from <was_home>\samples\src\PlantsByWebSphere\PlantsByWebSphereWEB\WebContent into the PlantsByWebSphereWeb/WebContent folder

Note: When you import source code or an EAR file, the workspace rebuilds the project. The build process runs as a separate background thread and can take a minute or two, depending on the application. In the lower right corner of the Application Server Toolkit window, a progress indicator tells you what is happening. When rebuild is complete, any error messages will appear in the Problems view.

If you click the **Problems** tab, you see that the workspace has a lot of problems. This is because the Web module depends on classes in the EJB module, but because we imported the source code, this meta data was not included, so we now need to set up the J2EE dependencies for the Web module to find its required classes.

1. In the Project Explorer, right-click the **PlantsByWebSphereWeb** project and select **Properties**.
2. Select the **J2EE Module Dependencies** section.
3. Select the **PlantsByWebSphereEJB.jar** option in the dialog. When selected, the Manifest Class-Path is updated so that the Web project references the EJB project. See Figure 13-5. Click **OK** when done.

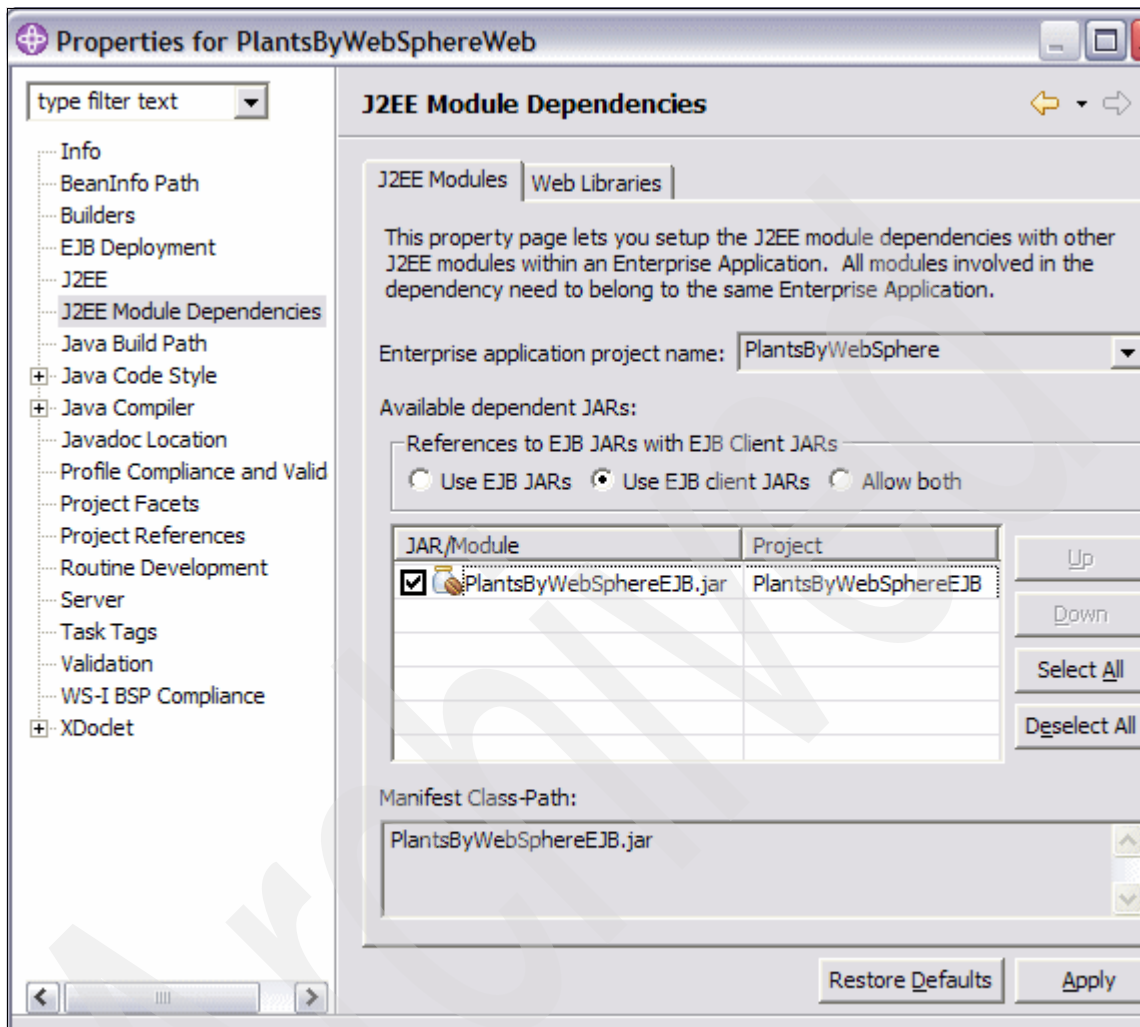


Figure 13-5 Configuring J2EE Module Dependencies

When the class path has been set up correctly and the project rebuilt, all errors in the Problems view should be gone. Still, there are a lot of warning messages.

None of these warnings are critical, though, so you do not need to bother about them. However, if you do want to get rid of them, you can select **Window** → **Preferences** → **Java** → **Compiler** → **Errors/Warnings**. Then expand the Unnecessary code section and change the settings for **Local variable never read** and **Unused local or private members** settings to **Ignore**.

The Application Server Toolkit will still warn you about a `serialVersionUID` field missing from several classes. This is because the toolkit now warns if a class is `Serializable` (implements `java.io.Serializable` interface) but does not have a line like:

```
private static final long serialVersionUID = 12345678L;
```

To fix this warning, you would add the above line to all serializable classes where it is missing. We will not do that here because the warning is not a severe one, so instead expand the **Potential programming problems** section and change the setting for **Serializable class without serialVersionUID** to **Ignore**. After a full rebuild of the workspace, there should be no warnings in the Problems view.

Note: The `serialVersionUID` is used for “version control” of serializable classes. Whenever you make a change to the class that is incompatible with the previous version, you should also change this number. Best practices dictates you should run the **serialver** utility (found in the JDK bin directory) to generate the number, but it is perfectly fine just to pick a number out of thin air. If the `serialVersionUID` line is not present, there is a small performance hit, because the JVM then needs to generate a number on the fly when the class is used.

Tip: When using the Application Server Toolkit, keep in mind the following:

- ▶ To perform a complete rebuild of your project(s), select **Project** → **Clean** and then select either to clean current or all projects. This will remove all build problems from the Problems view and perform a complete re-build of the selected projects. This sometimes removes errors and warnings in the Problems view.
- ▶ As you update and save modules in the toolkit, the contents of the modules are automatically validated and problems are listed in the Tasks view. You can also manually invoke validation of modules by selecting any module and choosing **Run Validation** from the context menu. To verify the settings for validation, select **Window** → **Preferences** and click **Validation**.

13.2.2 Working with deployment descriptors

Information describing a J2EE application and how to deploy it into a J2EE container is stored in XML files called deployment descriptors. An EAR file normally contains multiple deployment descriptors, depending on the modules it contains. Figure 13-6 shows a schematic overview of a J2EE EAR file. In this figure, the various deployment descriptors are designated with DD after their name.

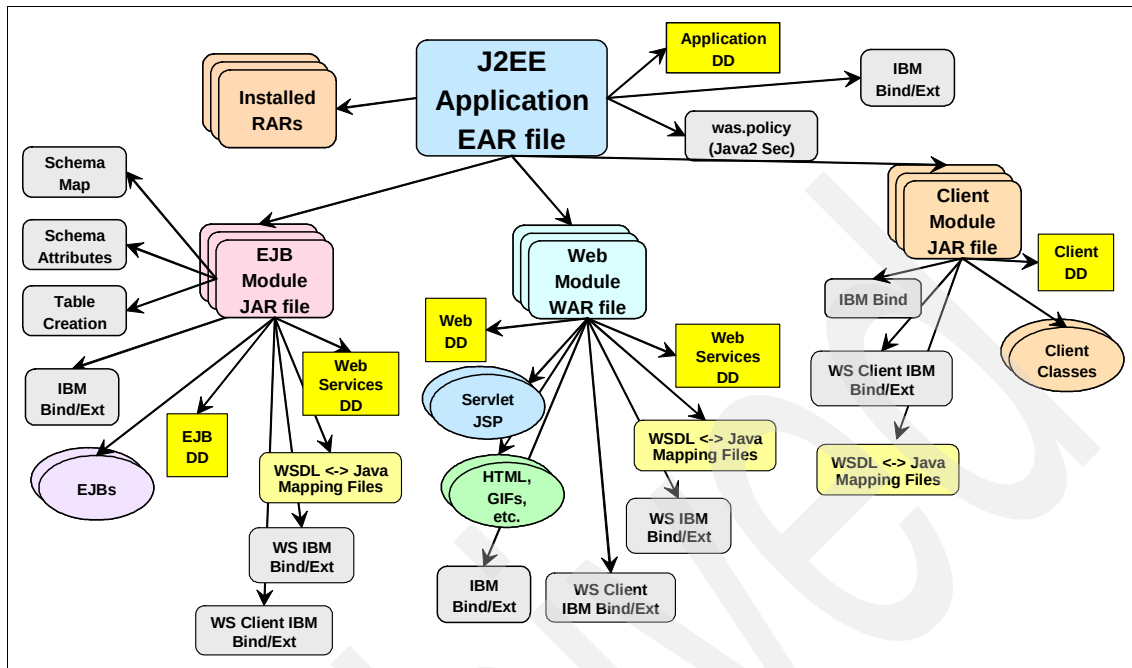


Figure 13-6 J2EE EAR file structure

The deployment descriptor of the EAR file itself is stored in the META-INF directory in the root of the EAR and is called application.xml. It contains information about the modules that make up the application.

The deployment descriptors for each module are stored in the META-INF directory of the module and are called web.xml (for Web modules), ejb-jar.xml (for EJB modules), ra.xml (for resource adapter modules), and application-client.xml (for application client modules). These files describe the contents of a module and allow the J2EE container to configure things like servlet mappings, JNDI names, and so forth.

Classpath information specifying which other modules and utility JARs are needed for a particular module to run is stored in the manifest.mf file also in the META-INF directory of the modules.

In addition to the standard J2EE deployment descriptors, EAR files produced by the Application Server Toolkit can also include additional WebSphere-specific information used when deploying applications to WebSphere environments. This supplemental information is stored in files called ibm-xxx-xxx-xxx.xmi, also in the META-INF directory of the respective modules. Examples of information in the

IBM-specific files are IBM extensions like servlet reloading and EJB access intents.

WebSphere Application Server V6.0 and V6.1 can also store deployment-related information (such as data sources, class loader settings, and so on) as part of an Enhanced EAR file. This information is stored in an `ibmconfig` subdirectory of the EAR file's `META-INF` directory.

The Application Server Toolkit has easy-to-use editors for working with all deployment descriptors. The information that goes into the different files are shown on one page in the GUI, eliminating the need to be concerned about what information is put into what file. However, if you are interested, you can click the Source tab of the deployment descriptor editor to see the text version of what is stored in that descriptor. For example, if you open the EJB deployment descriptor, you will see settings that are stored across multiple deployment descriptors for the EJB module, including:

- ▶ The EJB deployment descriptor, `ejb-jar.xml`
- ▶ The extensions deployment descriptor, `ibm-ejb-jar-ext.xmi`
- ▶ The bindings file, `ibm-ejb-jar-bnd.xmi` files
- ▶ The access intent settings, `ibm-ejb-access-bean.xmi`

To work with a deployment descriptor, do the following:

1. Open the J2EE perspective.
2. In the J2EE Project Explorer view, expand the project category (Enterprise Applications, EJB Projects, Dynamic Web Projects, and so on), and then expand the module you want to work with.
3. Double-click the deployment descriptor to open the editor for it. The deployment descriptor is module name with a version number in front. The version number refers to the J2EE specification level for the module. See Figure 13-7.

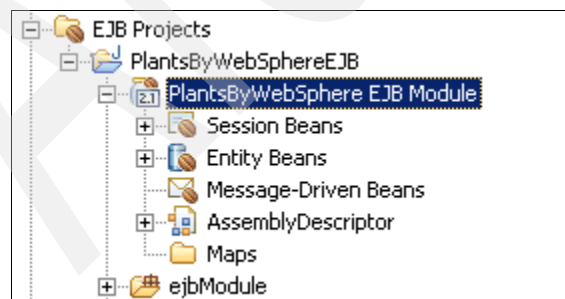


Figure 13-7 EJB deployment descriptor

4. Figure 13-8 on page 841 shows the deployment descriptor for the PlantsByWebSphere EJB module open with the deployment descriptor editor.

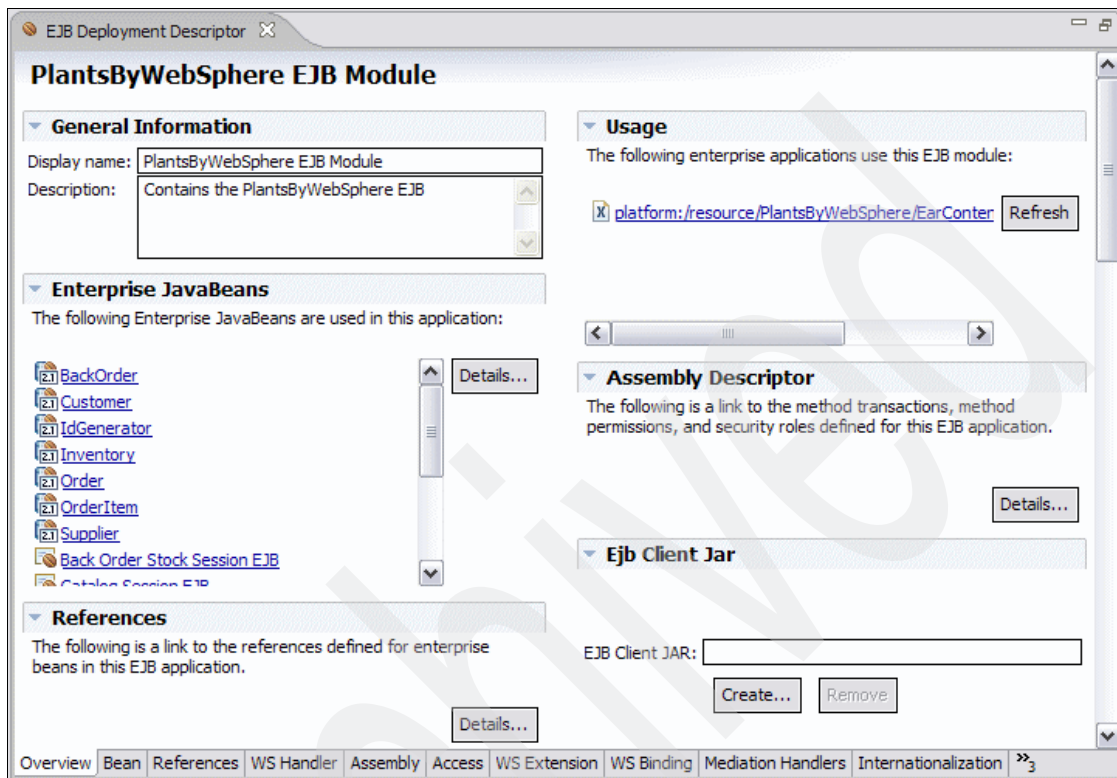


Figure 13-8 EJB deployment descriptor

The editor shows you information stored in all the relevant deployment descriptor files on the appropriate tabs. The descriptor files are kept in the META-INF directory of the module you are editing.

When you have made changes to a deployment descriptor, save it by pressing Ctrl+S and then close it.

13.3 Setting application bindings

At packaging time, you create references to resources. For an application to run, you need to bind these references to the real resources, such as JDBC data sources, created from the administrative console. This needs to be done for EJB references and resource references. You also need to define the enterprise bean's JNDI names, and security roles.

Bindings can be defined at development or deployment time. Most likely, developers will deliver a preconfigured EAR file that will then be modified at deployment time by the deployment team to suit the target environment.

All binding definitions are stored in the `ibm-xxx-bnd.xmi` files, where `xxx` can be `ejb-jar`, `Web`, `application`, or `application-client`.

In the next steps, you define the following bindings using the Application Server Toolkit:

- ▶ EJB JNDI names
- ▶ EJB references
- ▶ Data source for entity beans

All sections below assume that you have started the Application Server Toolkit and imported the Plants by WebSphere source, as described in “Import source code” on page 832.

13.3.1 Defining EJB JNDI names

For each session and entity bean, you must specify a JNDI name. This name is used to bind the EJB home object to an entry in the global JNDI name space. The bind happens automatically when the application server starts.

You can bind your EJBs anywhere you want in the JNDI name space, but best practice dictates they should be bound under the `ejb` subcontext, such as `ejb/Order` for the Order EJB. However, the Plants by WebSphere binds its EJBs to the `plantsby` subcontext in the JNDI root, so this is what we will show you here.

For clarity, however, when developing your own application, we recommend that you place all enterprise bean JNDI names for an application in a separate subcontext, such as `ejb/PlantsBy`. You can find the JNDI names for the Plants by WebSphere session and entity EJBs in Table 13-1.

Table 13-1 Plants by WebSphere enterprise bean JNDI names

EJB Name	JNDI Name
BackOrder entity bean	plantsby/BackOrderHome
Customer entity bean	plantsby/CustomerHome
IdGenerator entity bean	plantsby/IdGeneratorHome
Inventory entity bean	plantsby/InventoryHome
Order entity bean	plantsby/OrderHome
OrderItem entity bean	plantsby/OrderItemHome
Supplier entity bean	plantsby/SupplierHome
Back Order Stock Session EJB	plantsby/BackOrderStockHome
Catalog Session EJB	plantsby/CatalogHome
Login Session EJB	plantsby/LoginHome
Mailer Session EJB	plantsby/MailerHome
Report Generator Session EJB	plantsby/ReportGeneratorHome
Reset Database Session EJB	plantsby/ResetDBHome
Shopping Cart Session EJB	plantsby/ShoppingCartHome
Suppliers	plantsby/SuppliersHome

Use this table and the instructions below to define a JNDI name for each Plants by WebSphere enterprise bean:

1. In the Project Explorer view, expand the EJB Projects section.
2. Expand the **PlantsByWebSphereEJB** project and then double-click the Deployment Descriptor (**2.1 PlantsByWebSphere EJB Module**).

3. Click the **Bean** tab, and then click the **BackOrder EJB**. The EJB deployment descriptor is shown in Figure 13-9.
4. Look for the WebSphere Bindings section in the editor. In the JNDI name field, enter plantsby/BackOrderHome.

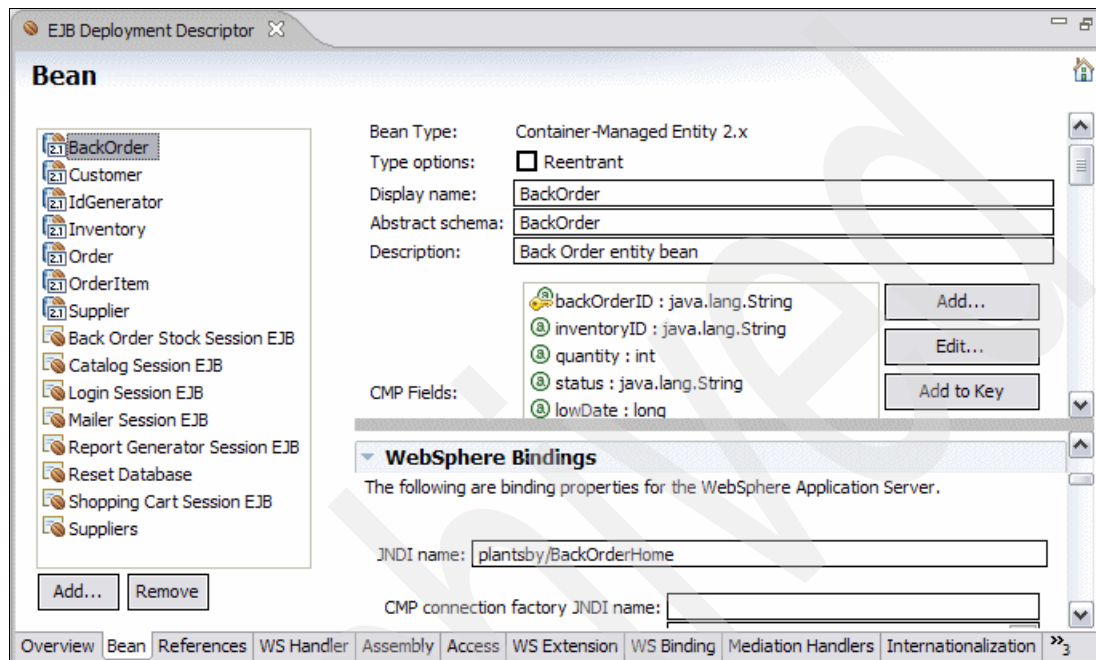


Figure 13-9 Defining EJB JNDI names

5. Repeat these steps for each of the session and entity enterprise beans in the EJB module.
6. Save the deployment descriptor.

13.3.2 Binding EJB and resource references

An EJB client can define *EJB references*, logical names, or nicknames, used by the client to find the EJB homes. When using references, the client can hard code the name of the reference. Then, during deployment, the reference is mapped to the real name in the JNDI namespace to which the EJB is bound. A reference to an EJB specifies either the local or remote home of the EJB.

For remote home interfaces, using EJB references is an option, but also a best practice. For local home interfaces, however, it is a must, because using an EJB reference is the only way an EJB client can look up a local home interface.

J2EE applications also use other kinds of references, such as resource references to look up data sources.

Table 13-2 on page 845 lists some of the EJB references used by the Plants by WebSphere sample.

Table 13-2 EJB and resource references: JNDI names list

EJB name	Reference name	JNDI Name
BackOrder	ejb/Inventory	plantsby/InventoryHome
BackOrder	ejb/IdGenerator	plantsby/IdGeneratorHome
Inventory	ejb/BackOrderStock	plantsby/BackOrderStockHome
Order	ejb/OrderItem	plantsby/OrderItemHome
Order	ejb/IdGenerator	plantsby/IdGeneratorHome
Login Session EJB	ejb/Customer	plantsby/CustomerHome

For example, to log on a user, the Login Session EJB uses the Customer EJB to find the user in a database. The Login Session EJB looks up the ejb/Customer entry in the JNDI name space. This is then mapped to plantsby/CustomerHome, which is the JNDI name the Customer EJB is bound to.

So the EJB reference is an alias used to find the target EJB. When using EJB references, the name of the reference must be prefixed with java:comp/env. So the Login Session EJB would do a JNDI lookup with the name java:comp/env/ejb/Customer to get the EJB reference for Customer.

Follow these steps to bind an EJB to a JNDI name:

1. In the EJB Deployment Descriptor for the PlantsByWebSphereEJB module (see Figure 13-9 on page 844), click the **References** tab.

2. Click the plus sign next to the Login Session EJB and select the **ejb/Customer** reference, as shown in Figure 13-10.

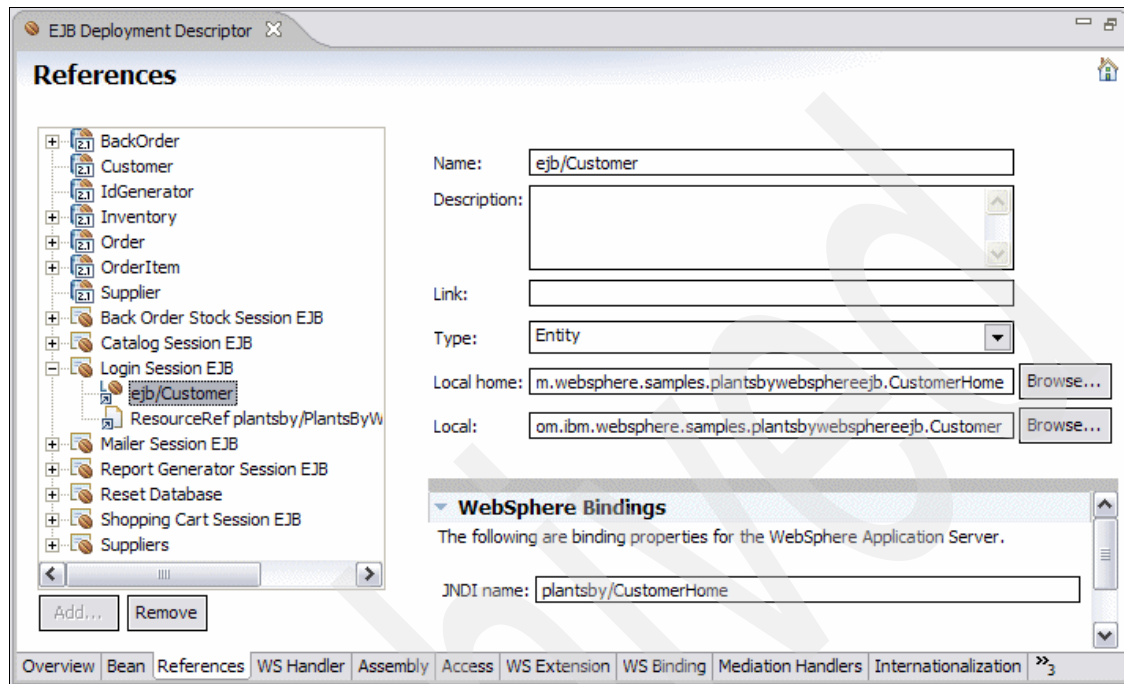


Figure 13-10 Setting EJB References bindings

3. In the WebSphere bindings section, specify plantsby/CustomerHome.
4. Save the deployment descriptor.

Note: EJB reference bindings can be defined or overridden at deployment time in the administrative console for all modules except for application clients, for which you must use the Application Server Toolkit.

13.3.3 Defining data sources for entity beans

The entity beans in the Plants by WebSphere application are container-managed (CMP) EJBs. The EJB container handles the persistence of the EJB attributes in the underlying persistent store. You must specify which data store to use. This is done by binding an EJB module or an individual EJB to a data source. If you bind the EJB module to a data source, all EJBs in that module use the same data source for persistence. If you specify the data source at the EJB level, then this data source is used instead.

For the Plants by WebSphere application, the data source is bound at the EJB module level. The data source configured for the EJB must match the data source configured in the WebSphere environment. The JNDI name for this data source is `eis/jdbc/PlantsByWebSphereDataSource_CMP`. When a data source is defined in WebSphere, it can be marked as for use by container-manager persistence. If this option is selected the data source is then also mapped into the JNDI name space with a name like this.

To bind the Plants by WebSphere EJB module to this data source, follow these steps:

1. In the EJB Deployment Descriptor for the `PlantsByWebSphereEJB` module (see Figure 13-9 on page 844), click the **Overview** tab.
2. In the Overview tab, scroll down and find the WebSphere bindings section, as in Figure 13-11.

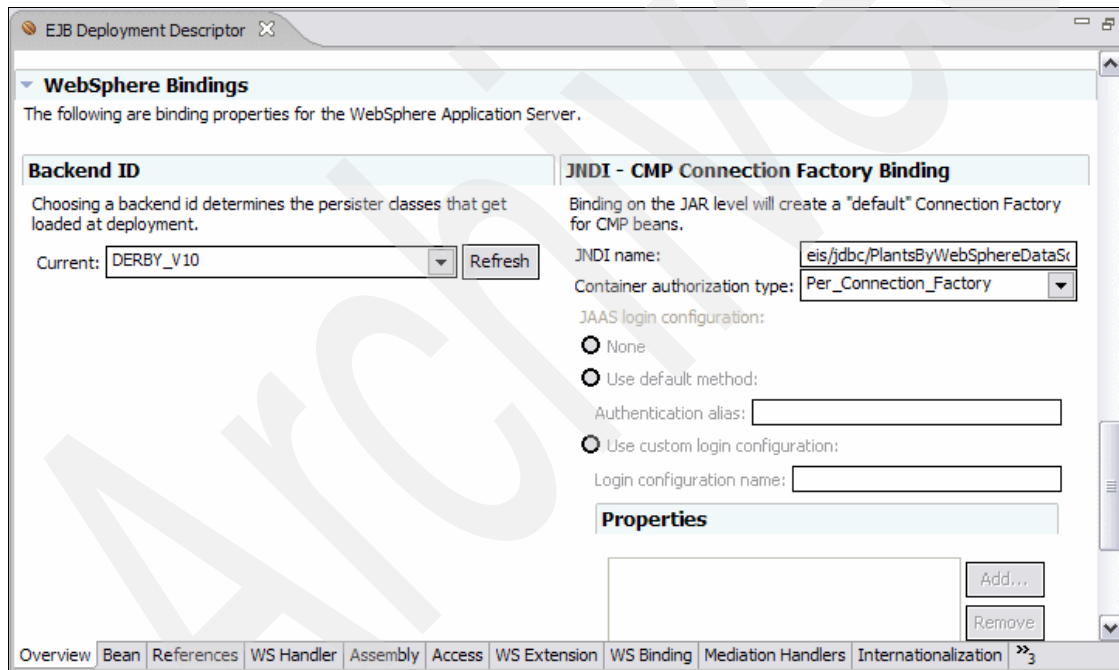


Figure 13-11 Specifying the default CMP data source for the EJB module

Check the following fields:

New in V6.1: The embedded Cloudscape has a Cloudscape v10.1.x code base, referred to as Derby. Derby is a product of the Apache Software Foundation (ASF) open source relational database project. The new Cloudscape includes Derby without any modification to the underlying source code. Learn more about Derby code at the Apache Derby Web site: <http://db.apache.org/derby/>

- Backend ID

In EJB 2.x, mapping and schema files make up a back end for EJB 2.x projects. The Plants by WebSphere sample ships with a Cloudscape back end defined. Leave this for now, though in “Creating a new database mapping and schema” on page 848, we will create a new back end for DB2 and change this binding to point it.

- JNDI name

Enter `eis/jdbc/PlantsByWebSphereDataSource_CMP` in the JNDI name field. This is the value the application uses to access the database. Note that this is the same, regardless of which back-end ID is used.

- Container authorization type

Select **Per_Connection_Factory** for the Container authorization type.

3. Save the deployment descriptor.

Tip: An EJB JAR can contain database mappings and EJB deployment code for multiple databases. Because we imported the source code for the Plants by WebSphere sample, there is currently no deployed code, but we will generate it for DB2 UDB. You can set which back-end ID will be used at run time in the WebSphere bindings section. This choice can also be overridden at deployment time.

Creating a new database mapping and schema

The Plants by WebSphere sample is configured to run against a Cloudscape database. However, for the purpose of showing how to create a new database back end and deployed code, we will configure it so it can also run against a DB2 database.

Creating the database mapping

First, you need to create a database mapping using the EJB project. To do this, perform the following steps:

1. Expand the **EJB Projects** section in the Project Navigator view and then expand the **PlantsByWebSphereEJB** project.
2. Right-click the deployment descriptor and select **EJB to RDB Mapping** → **Generate Map**.
3. Select **Create a new backend folder** and click **Next**.
4. Select **Top-down** and click **Next**.
5. On the Top-down mapping options page, select **DB2 UDB V8.2**, or the corresponding DB2 product and version you are using as the target database. Enter database name **PLANTS** and leave **NULLID** as the schema name. See Figure 13-12 on page 849.

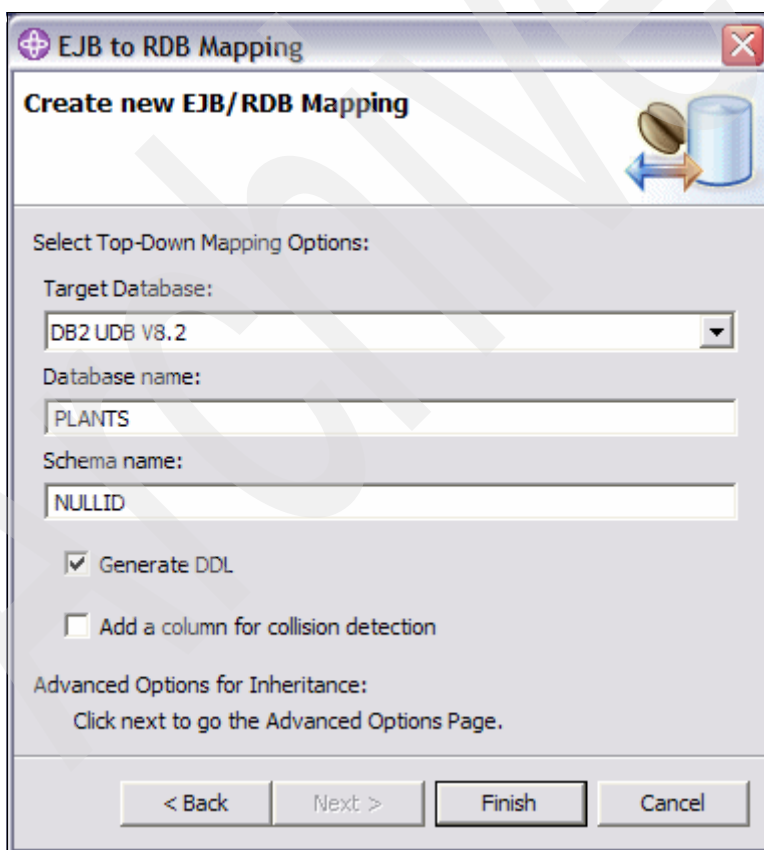


Figure 13-12 Generating a DB2 mapping

6. Click **Finish**.

The Application Server Toolkit database mapping editor (Map.mapxmi editor) opens, allowing you to make adjustments to the mapping between the fields of the entity EJBs and the database columns. We do not need to do that, so close the editor.

A Table.ddl file containing the script to set up the DB2 tables is created in the PlantsByWebSphereEJB\ejbModule\META-INF\backends\DB2UDBNT_V82_1 directory of the Application Server Toolkit workspace. You will need this script when creating the DB2 PLANTS database before deploying Plants by WebSphere.

The sample application stores images of the products you can buy in the INVENTORY table in the database. The field of the EJB that holds the image data is declared as a byte array (byte[]) and the Application Server Toolkit maps this to a DB2 type of VARCHAR (1000) FOR BIT DATA (as you can see if you open the database mapping editor on Map.mapxmi). We did not get this data type to work, so we changed it to a 100 KB large BLOB instead. To do this, you need to open the Database Table Editor. If you are familiar with the Application Server Toolkit or Rational Application Developer, you should pay attention, because in the Application Server Toolkit V6.1, it is not accessed in the same way.

7. Double-click the PLANTS.dbm file in the PlantsByWebSphereEJB\ejbModule\META-INF\backends\DB2UDBNT_V82_1 folder.
8. Click the plus sign that appears in front of PLANTS.dbm and expand the tree all the way down to the INVENTORY table and then expand the INVENTORY table as well.
9. Select the IMGBYTES column.
10. In the Properties view, click the **Type** tab, as shown Figure 13-13.

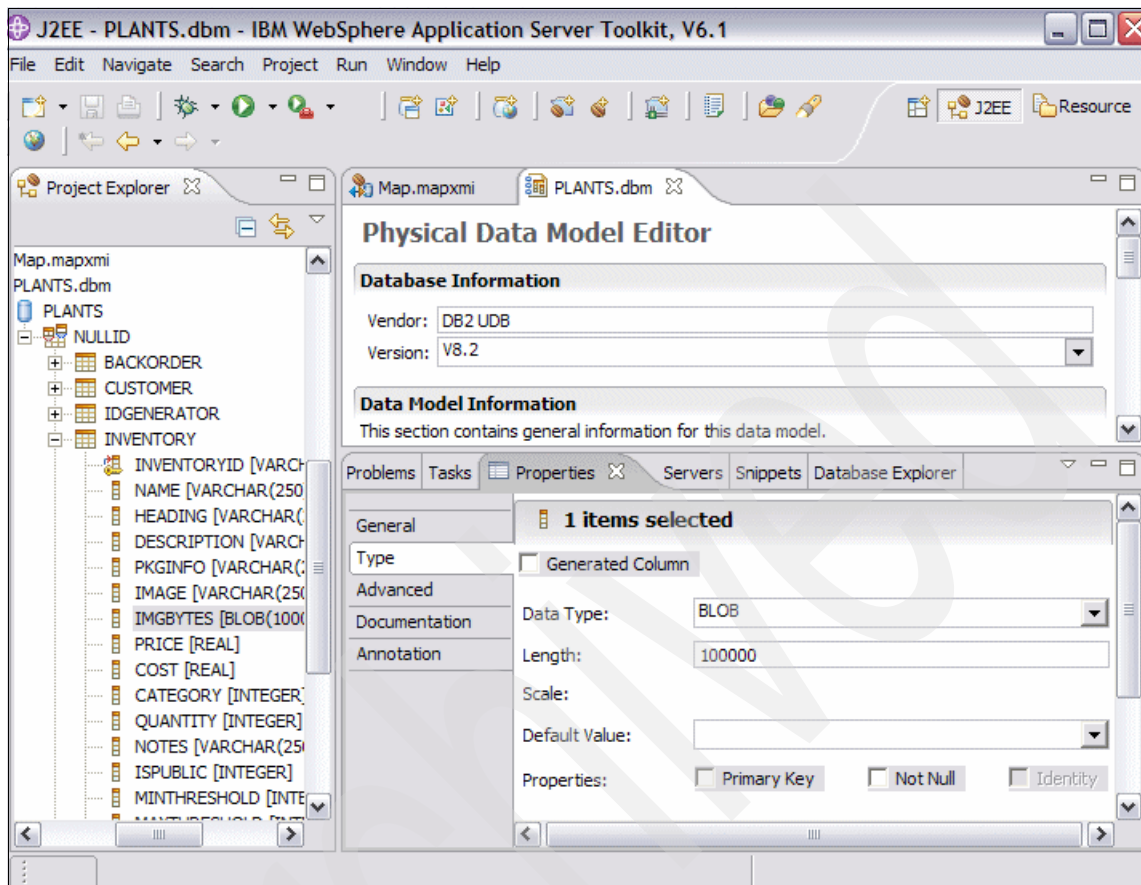


Figure 13-13 Table Editor

11. For data type, select **BLOB**. Then enter 100000 in the length field. Press Ctrl-S to save the change to the PLANTS.dbm file.
12. Because the mapping now has changed, we need to update the Table.ddl file. Right-click the **Deployment Descriptor (2.1 PlantsByWebSphere EJB Module)** project and select **EJB to RDB Mapping → Generate Schema DDL**.

13. The Table.ddl file in the PlantsByWebSphereEJB\ejbModule\META-INF\backends\DB2UDBNT_V82_1 folder is now updated. Example 13-1 shows part of the file.

Example 13-1 Plants by WebSphere Table.ddl

```
CREATE TABLE CUSTOMER (  
    CUSTOMERID VARCHAR(250) NOT NULL,  
    PASSWORD1 VARCHAR(250),  
    FIRSTNAME VARCHAR(250),  
    LASTNAME VARCHAR(250),  
    ADDR1 VARCHAR(250),  
    ADDR2 VARCHAR(250),  
    ADDR_CITY VARCHAR(250),  
    ADDR_STATE VARCHAR(250),  
    ADDR_ZIP VARCHAR(250),  
    PHONE VARCHAR(250)  
);  
CREATE TABLE INVENTORY (  
    INVENTORYID VARCHAR(250) NOT NULL,  
    NAME VARCHAR(250),  
    HEADING VARCHAR(250),  
    DESCRIPTION VARCHAR(250),  
    PKGINFO VARCHAR(250),  
    IMAGE VARCHAR(250),  
    IMGBYTES BLOB(100000),  
    PRICE REAL NOT NULL,  
    COST REAL NOT NULL,  
    CATEGORY INTEGER NOT NULL,  
    QUANTITY INTEGER NOT NULL,  
    NOTES VARCHAR(250),  
    ISPUBLIC INTEGER NOT NULL,  
    MIN_THRESHOLD INTEGER NOT NULL,  
    MAX_THRESHOLD INTEGER NOT NULL  
);  
...  
...  
ALTER TABLE CUSTOMER ADD CONSTRAINT PK_CUSTOMER PRIMARY KEY  
(CUSTOMERID);  
ALTER TABLE INVENTORY ADD CONSTRAINT PK_INVENTORY PRIMARY KEY  
(INVENTORYID);
```

14. We can now generate the EJB deployment code necessary to support our DB2 database. Right-click the deployment descriptor for the PlantsByWebSphere EJB module and select **Deploy** to generate the EJB deployment code.

If you get any errors in the Problems view, you can try a Clean operation, as this sometimes eliminates them.

Change the back-end ID

Because we have created a new database back-end map, we can set the default back-end map for the EJB to the newly created DB2 map. To map to the new DB2 map, do the following:

1. Open the deployment descriptor for the EJB module, scroll down to the bottom of the Overview tab and select **DB2UDBNT_V82_1** as the Current Backend ID, as shown in Figure 13-14.

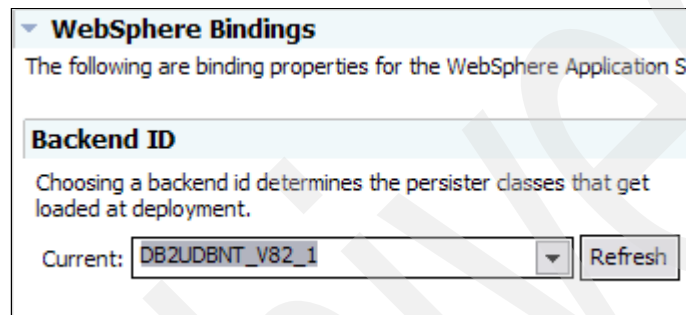


Figure 13-14 Setting default backend id for EAR file

2. Press Ctrl-S to save the deployment descriptor.

13.3.4 Setting the context root for Web modules

The context root is the part of the URL that comes after the protocol and address but before the path to servlets, JSPs, and so on. In a URL such as <http://www.plantsbywebsphere.com/PlantsByWebSphere/servlet/ShoppingServlet>, the context root is **PlantsByWebSphere**. It is used to separate Web modules from each other so that the Web container can dispatch Web requests to the right Web module.

When we created the **PlantsByWebSphereWeb** module, the Application Server Toolkit assigned it a context root with the same path, **PlantsByWebSphereWeb**. Because the sample application shipped and installed with WebSphere (if you install the samples) uses a context root of **PlantsByWebSphere**, we will change the context root also for our module to match.

1. Expand the Dynamic Web Projects section. Right-click the **PlantsByWebSphereWeb** module and select **Properties**.
2. Select the J2EE tab.

3. Change the context root to PlantsByWebSphere, as shown in Figure 13-15.

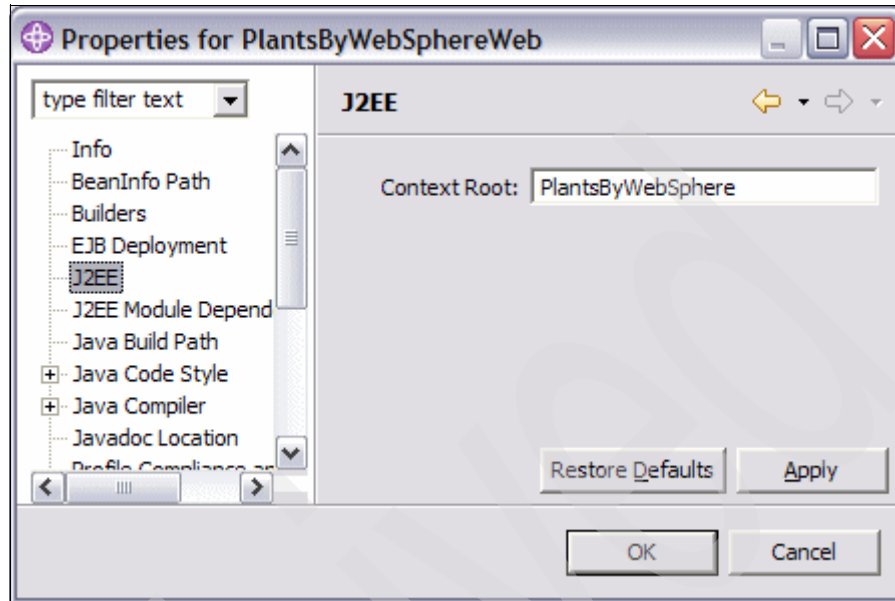


Figure 13-15 Setting context root for Web module

Click **OK**.

13.4 IBM EJB extensions: EJB caching options

This section discusses the caching options for entity and stateful session beans.

13.4.1 EJB container caching option for entity beans

The Enterprise JavaBeans specification defines three EJB caching options: options A, B, or C. Those options define how the EJB container handles entity bean instances between transactions. EJB caching options are set at the bean level, and are part of the IBM extensions deployment descriptor.

Caching option A

With caching option A, you assume that the entity bean has *exclusive* access to the underlying persistent store. In other words, between transactions, no one will modify the data. This includes a batch program updating the data, a Java application updating the data, or even the same entity bean running in a different container. This implies option A cannot be used in a clustered environment

(WLM). Note that it is your responsibility to ensure no other application will modify the data, as the EJB container has no way to control write access to the underlying database from other servers.

When caching option A is used, the entity bean instance is kept in a memory cache across transactions. At transaction commit, the entity bean attributes are synchronized with the underlying persistent store, and the bean instance remains cached in memory.

If you were tracing the calls made by the container, you would see something similar to Example 13-2. The first time the entity bean is used, its run time context is set (step 1), a bean is taken from the entity beans instance pool (step 2), the bean instance attributes are synchronized with the underlying data store (step 3), the method `setBalance` is invoked on the bean (step 4), and, finally, the bean attributes are saved back to the database (step 5). The bean is not returned to the pool. On subsequent calls, the `setBalance` method is invoked directly on the cached bean instance, and the bean attributes are synchronized with the underlying persistent data store.

Example 13-2 Entity beans call trace with option A caching

Transaction 1 (Begin)

Step 1: 1c9585f1 BranchAccount E called `setEntityContext()` method

Step 2: 1c9585f1 BranchAccount E called `ejbActivate()` method

Step 3: 1c9585f1 BranchAccount E called `ejbLoad()` method

Step 4: 1c9585f1 BranchAccount E called `setBalance()` method

Step 5: 1c9585f1 BranchAccount E called `ejbStore()` method

Transaction 1 (Commit)**Transaction 2 (Begin)**

Step 1: 284485f1 BranchAccount E called `setBalance()` method

Step 2: 284485f1 BranchAccount E called `ejbStore()` method

Transaction 2 (Commit)

Using caching option A can provide some performance enhancements at the expense of higher memory usage. You should only use it if you do not intend to use WebSphere clustering capabilities and you mostly access data in read mode.

Caching option B

With caching option B, you assume that you have *shared* access to the underlying database. This means the data could be changed by another application between transactions. When option B is used, the bean instance attributes are always synchronized with the underlying back-end data store at the beginning of every transaction. Similar to Option A, the bean is kept in the cache

between transactions. Therefore, if you were tracing the different calls made in Option B, you would obtain the trace shown in Example 13-3.

Example 13-3 Entity beans call trace with option B caching

Transaction 1 (Begin)

Step 1: 1c9585f1 BranchAccount E called setEntityContext() method

Step 2: 1c9585f1 BranchAccount E called ejbActivate() method

Step 3: 1c9585f1 BranchAccount E called ejbLoad() method

Step 4: 1c9585f1 BranchAccount E called setBalance() method

Step 5: 1c9585f1 BranchAccount E called ejbStore() method

Transaction 1 (Commit)

Transaction 2 (Begin)

Step 1: 284485f1 BranchAccount E called ejbLoad() method

Step 2: 284485f1 BranchAccount E called setBalance() method

Step 3: 284485f1 BranchAccount E called ejbStore() method

Transaction 2(Commit)

Caching option B can be safely used in a clustered environment, or when you are not sure if you have exclusive access to data. You are assured that you always work with the last committed data. Option B memory usage is the same as for option A. The performance of both options can slightly differ depending on the nature of your application.

Caching option C

Similar to option B, caching option C assumes *shared* access to the database. Unlike option B or A, the bean instance is returned to the entity beans pool at the end of the transaction. A new bean instance is used at the beginning of every transaction. Each transaction results in the sequence of calls shown in Example 13-4.

Example 13-4 Entity beans call trace with option C caching

Transaction (Begin)

Step 1: 1c9585f1 BranchAccount E called setEntityContext() method

Step 2: 1c9585f1 BranchAccount E called ejbActivate() method

Step 3: 1c9585f1 BranchAccount E called ejbLoad() method

Step 4: 1c9585f1 BranchAccount E called setBalance() method

Step 5: 1c9585f1 BranchAccount E called ejbStore() method

Step 6: 1c9585f1 BranchAccount E called ejbPassivate() method

Step 7: 1c9585f1 BranchAccount E called unsetEntityContext() method

Transaction (Commit)

Caching option C has the best memory usage at the expense of a larger number of methods calls. This is the default behavior.

How to set the EJB caching option

You must combine the Activate at and Load at options to set the EJB caching option to A, B, or C. Use Table 13-3 to choose the right combination.

Table 13-3 Setting entity EJB caching properties

Option	Activate at must be set to	Load at must be set to
Option A	Once	Activation
Option B	Once	Transaction
Option C (default)	Transaction	Transaction

To set the EJB caching option, do the following:

1. Open the EJB deployment descriptor.
2. Switch to the **Bean** tab.
3. Select the entity bean in the window to the left, then scroll down the options at right until you see the Bean Cache settings under the WebSphere extensions section, as in Figure 13-16.

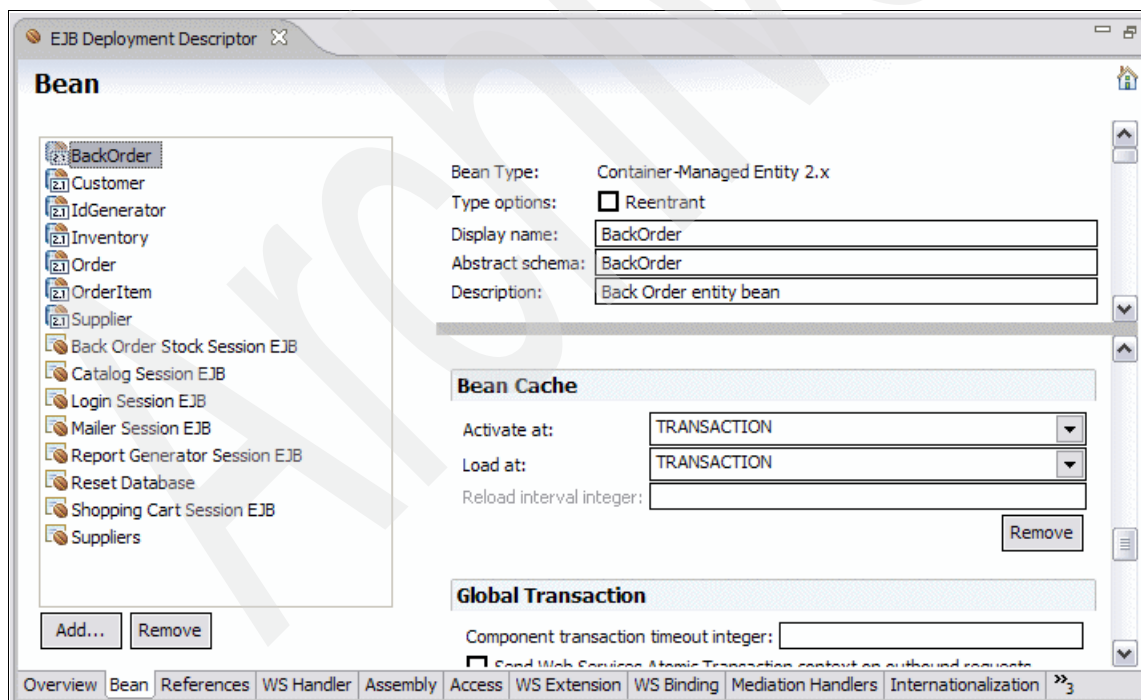


Figure 13-16 Setting the activate and load settings for entity beans

4. Select the **Activate at** and **Load at** options according to Table 13-3 on page 857.
5. Close and save the deployment descriptor.

The settings are saved in the `ibm-ejb-jar-ext.xml` file. They correspond to the following line:

```
<beanCache xmi:id="BeanCache_1" activateAt="ONCE" loadAt="TRANSACTION"/>
```

There is one line for each entity bean for which you have set this option.

In addition to the two standard options for EJB activation and data loading, WebSphere also supports some additional options, such as activate at activity session and to load data on a daily, weekly, or other interval basis. For more information about these WebSphere extensions, refer to the WebSphere IntoCenter (search for bean cache settings).

Note: Although the Application Server Toolkit and Rational Application Developer allow you to set the Activate at and Load at options also for stateless session EJBs, they have no effect.

13.4.2 EJB container caching option for stateful session beans

Similarly to entity beans, you can specify which caching strategy to use for stateful session beans. This caching option specifies the point at which an enterprise bean is activated and placed in the cache. Removal from the cache and passivation are also governed by this setting. Valid values are:

- ▶ Once (default)
Choosing Once indicates that the bean is activated when it is first accessed in the server process. It is passivated and removed from the cache at the discretion of the container, for example, when the cache becomes full.
- ▶ Transaction
Choosing Transaction indicates that the bean is activated at the start of a transaction. It is passivated and removed from the cache at the end of the transaction.

You can set this caching option by opening the EJB deployment descriptor for the EJB module. The Activate at setting is found on the Bean tab (Figure 13-17). Select the bean and scroll down to the Bean Cache category.

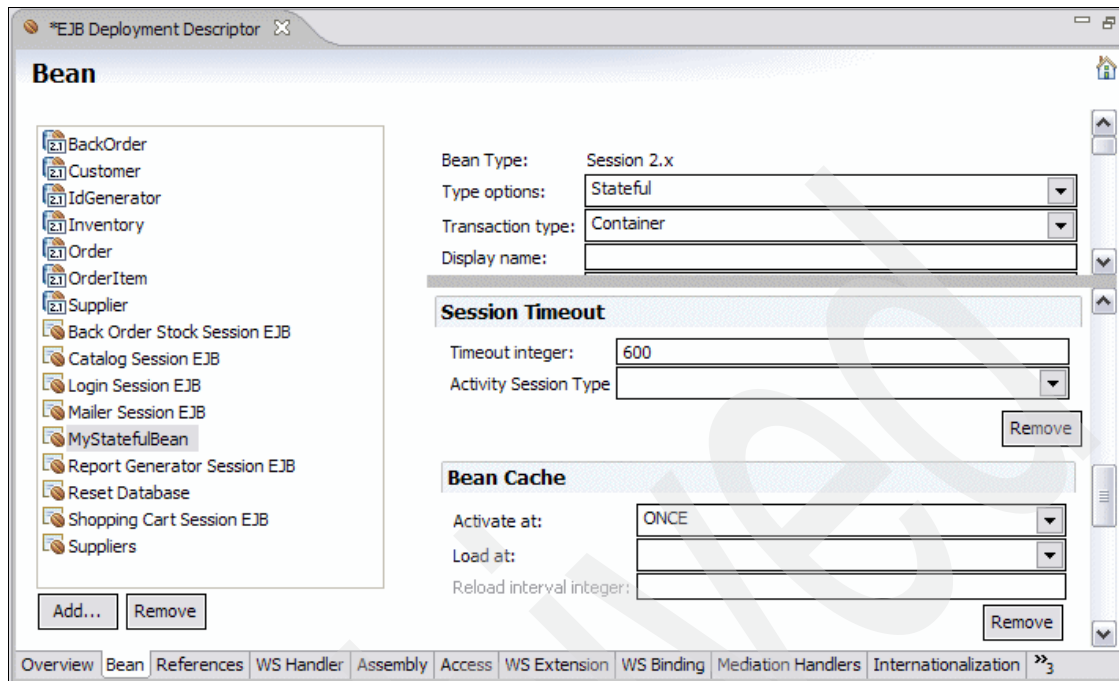


Figure 13-17 Activate settings for stateful session beans

Note: The Load at option does not have any effect on Stateful Session EJBs.

13.4.3 Stateful EJB timeout option

Additionally, you can specify a timeout value for stateful session beans. A bean can time out in the METHOD_READY or in the PASSIVATED state. If you try to access a bean that has timed out, you see an exception similar to that in Example 13-5.

Example 13-5 Stateful EJB timed out exception

```
com.ibm.ejs.container.SessionBeanTimeoutException: Stateful bean
StatefulBean0(BeanId(Webbank#webbankEJBs.jar#Transfer, ebf64d846a), state =
METHOD_READY) timed out.
```

Session beans that have timed out can be removed by the container, for example, if it needs to free memory. However, a well-written application should not rely on beans to time out to free memory. Instead, it is important that the

developer explicitly calls `remove()` on a bean when this stateful bean is not needed anymore.

The default timeout is 600 seconds. You can set the timeout integer value as a parameter of a stateful session bean by opening the EJB deployment descriptor and selecting the **Bean** tab, as in Figure 13-17 on page 859. By specifying a value of 0, you set the bean to never expire.

Setting this timeout inserts the following property in the `ejbExtensions` tag of the IBM bindings file:

```
<ejbExtensions xmi:type="ejbext:SessionExtension"
  xmi:id="Session_1_Ext" timeout="120">
```

Note: If a bean times out in the `METHOD_READY` state and is consequently removed by the container, the `ejbRemove()` method is called on the bean instance. If a bean times out in the passivated state, `ejbRemove()` is not called, according to the EJB specification.

13.5 IBM EJB extensions: EJB access intents

Access intents are used to optimize the access to relational data. Access intents are only applicable to EJB 2.x beans. For EJB 1.1 beans, you still use the old method of setting the transaction isolation level at the method level, as well as mark methods as read-only. In this section, we only cover access intents for EJB 2.x beans.

Access intent policies are specifically designed to supplant the use of isolation level and read-only, method-level modifiers found in the extended deployment descriptor for EJB Version 1.1 enterprise beans. You cannot specify isolation level and read-only modifiers for EJB Version 2.x enterprise beans. The WebSphere persistence manager uses access intent hints to make decisions about isolation level, cursor management, and more.

13.5.1 Transaction isolation levels overview

Transaction isolation levels provide a trade-off between accuracy of reads versus concurrent readers. The levels can best be described by the types of read anomalies they permit and forbid. Consider the read anomalies that can occur with two concurrent transactions, T1 and T2:

- Dirty read

T1 reads data that has been modified by T2, before T2 commits.

► Non-repeatable read

Non-repeatable read is caused by fine-grained locks.

- T1 reads a record and drops its lock.
- T2 updates.
- T1 re-reads different data.

► Phantom read

This is a non-repeatable read involving a range of data and inserts or deletes on the range.

- T1 reads a set of records that match some criterion.
- T2 inserts a record that matches the criterion.
- T1 continues processing the set, which now includes records that were not part of the original matching set.

There are four possible settings for the transaction isolation level:

► Repeatable read (TRANSACTION_REPEATABLE_READ)

This setting permits phantom reads and forbids both dirty and unrepeatable reads.

► Read committed (TRANSACTION_READ_COMMITTED)

This setting permits non-repeatable and phantom reads and forbids dirty reads.

► Read uncommitted (TRANSACTION_READ_UNCOMMITTED)

This setting permits all the read anomalies, including dirty reads, non-repeatable reads, and phantom reads.

► Serializable (TRANSACTION_SERIALIZABLE)

This setting forbids all the read anomalies.

The container applies the isolation level as follows:

- For entity beans with container managed persistence (CMP), the container generates code that ensures the desired level of isolation for each database access.
- For session beans and bean-managed persistence (BMP) entity beans, the container sets the isolation level at the start of each transaction, for each database connection.

The transaction isolation level is tied to a database connection. The connection uses the isolation level specified in the first bean that uses the connection. The

container throws an `IsolationLevelChangeException` whenever the connection is used by another bean method that has a different isolation level.

Not all databases support all JDBC isolation levels. Moreover, JDBC definitions for isolation levels might not match the database definition of isolation levels. As an example, DB2 definitions for isolation levels follow the naming conventions used in *Transaction Processing: Concepts and Techniques* by Gray. Table 13-4 shows a mapping between EJB and DB2 isolation levels.

Table 13-4 Mapping JDBC isolation levels to DB2 isolation levels

JDBC isolation level	DB2 isolation level
TRANSACTION_SERIALIZABLE	Repeatable Read
TRANSACTION_REPEATABLE_READ	Read Stability
TRANSACTION_READ_COMMITTED	Cursor Stability
TRANSACTION_READ_UNCOMMITTED	Uncommitted Read

To learn more, refer to the documentation provided by your database product.

13.5.2 Concurrency control

Concurrency control is the management of contention for data resources. A concurrency control scheme is considered *pessimistic* when it locks a given resource early in the data-access transaction and does not release it until the transaction is closed. A concurrency control scheme is considered *optimistic* when locks are acquired and released over a very short period of time at the end of a transaction.

The objective of optimistic concurrency is to minimize the time over which given resource is unavailable for use by other transactions. This is especially important with long-running transactions, which under a pessimistic scheme would lock up a resource for unacceptably long periods of time.

WebSphere uses *an overqualified update scheme* to test whether the underlying data source has been updated by another transaction since the beginning of the current transaction. With this scheme, the columns marked for update and their original values are added explicitly through a WHERE clause in the UPDATE statement so that the statement fails if the underlying column values have been changed. As a result, this scheme can provide column-level concurrency control; pessimistic schemes can control concurrency at the row level only.

Optimistic schemes typically perform this type of test only at the end of a transaction. If the underlying columns have not been updated since the

beginning of the transaction, pending updates to container-managed persistence fields are committed and the locks are released. If locks cannot be acquired or if some other transaction has updated the columns since the beginning of the current transaction, the transaction is rolled back and all work performed within the transaction is lost.

Pessimistic and optimistic concurrency schemes require different transaction isolation levels. Enterprise beans that participate in the same transaction and require different concurrency control schemes cannot operate on the same underlying data connection unless the connection is able to change its isolation level on an individual-query basis. Some, but not all, JDBC drivers can do this. For those JDBC drivers that cannot, mixing concurrency controls requires the use of multiple connections within a transaction.

Whether or not to use optimistic concurrency depends on the type of transaction. Transactions with a high penalty for failure might be better managed with a pessimistic scheme. For low-penalty transactions, it is often worth the risk of failure to gain efficiency through the use of an optimistic scheme.

13.5.3 Using EJB 2.x access intents

Access intents policies let you define, in a very flexible and powerful way, how relational data will be accessed if you use BMP or CMP entity beans.

Important: Although you can set access intents on a BMP, you are responsible for reading the access intent metadata from your code and applying the corresponding change to the isolation levels yourself by calling `connection.setTransactionLevel()`. The EJB container has no control over your persistence strategy, and therefore cannot perform the same tasks as it can for CMPs. Access intent data is valid in the WebSphere naming service for the time of the transaction. See the Information Center for more coding examples.

Access intents policies

Seven access intent policies are available. They cover a wide variety of ways to access data. They are summarized in Table 13-5 on page 864.

Table 13-5 Access intent policies

Access Intent Policy	Concurrency control	Used for update	Transaction isolation level	Notes
wsPessimisticRead	Pessimistic	No	For Oracle, read committed. Otherwise, repeatable read.	Read locks are held for the duration of the transaction. Updates are not permitted; the generated SELECT query does not include FOR UPDATE.
wsPessimisticUpdate	Pessimistic	Yes	For Oracle, read committed. Otherwise, repeatable read.	The generated SELECT FOR UPDATE query grabs locks at the beginning of the transaction.
wsPessimisticUpdate-Exclusive	Pessimistic	Yes	Serializable.	SELECT FOR UPDATE is generated; locks are held for the duration of the transaction.
wsPessimisticUpdate-NoCollision	Pessimistic	Yes	Read committed.	The generated SELECT query does not include FOR UPDATE. <i>No locks are held, but updates are permitted.</i>
wsPessimisticUpdate-WeakestLockAtLoad (DEFAULT VALUE)	Pessimistic	Yes	For Oracle, read committed, Otherwise, repeatable read.	For Oracle, this is the same as wsPessimisticUpdate. Otherwise, the generated SELECT query does not include FOR UPDATE; locks are escalated by the persistent store at storage time if updates were made.
wsOptimisticRead	Optimistic	No	Read committed.	

Access Intent Policy	Concurrency control	Used for update	Transaction isolation level	Notes
wsOptimisticUpdate	Optimistic	Yes.	Read committed.	Generated overqualified-update query forces failure if CMP column values have changed since the beginning of the transaction.

There are two critical questions to ask yourself when using access intents:

- ▶ At which point in my transaction do I access data? This is critical in selecting on which method you must set the access intent.
- ▶ How do I want to access data? This is critical to selecting the best access intent to apply on the method.

Choosing where to apply the access intent

This is critical because it is at this point that the WebSphere persistence manager decides which access intent to use. To illustrate this point, we will use the following example.

Assume a session EJB called Consultation obtains the CustomerAccount balance using `getBranchAccountBalance`, as in Example 13-6.

Example 13-6 Obtaining CustomerAccount balance using getBranchAccountBalance

```
int getCustomerAccountBalance () {
    ...
    custAcct = (CustomerAccountLocal) custAcctHome.findByPrimaryKey(custAcctKey);
    return custAcct.getBranchBalance();
}
```

Imagine you have applied `AccessWriteIntent1` on the `findByPrimaryKey()` method of the `CustomerAccount` bean, and `AccessReadIntent2` on the `getBranchBalance()` method. Because the first access to the database in the transaction started by the call to `getCustomerAccountBalance()` is done by the `findByPrimaryKey` method, then `AccessWriteIntent1` is used for all calls within the transaction. `AccessReadIntent2` will be ignored by the persistence manager and is therefore useless in this case.

This might be satisfactory or not, depending on what you want to achieve. The critical point here is that you can use the `findByPrimaryKey` method in read and update transactions. If you use it in an update transaction, you probably want to

execute it with, for example, a `PessimisticUpdate` intent. If you access data only for reading it, this would be more than you need.

There are two main solutions you can adopt for this problem. The simplest one is to have two or more versions of your finder methods, specialized by access intent. You could use the standard `findByPrimaryKey` in update scenarios and add another finder method, such as `findByPrimaryKeyForRead`, and use it in read-only scenarios. You would set the access intent of the `findByPrimaryKeyForRead` finder, say to `wsOptimisticRead`. The default access intent (`wsPessimisticUpdate-WeakestLockAtLoad`) is fine in most cases for the `findByPrimaryKey()` method, as well as for other finder or non-finder methods.

Important: This solution is also well adapted to BMPs. By having a different `findByPrimaryKey` method for read and write transactions, you can easily set a different isolation level in the code for each of them. You can also define a different `SELECT` query, one with a `FOR UPDATE` clause and one without, then call them from those different methods.

Another solution is to run `findByPrimaryKey`, or another finder, in its own transaction. Do this by applying a `RequiresNew` transaction flag on it. Take another look at the previous sample:

1. The `getCustomerAccountBalance` method starts a new transaction.
2. `findByPrimaryKey` is called. The current transaction is paused. The `findByPrimaryKey` method executes within its own transaction and, therefore, own access intent. The call to `findByPrimaryKey()` returns an *unhydrated* instance, which means it has not been activated nor loaded.
3. The transaction initiated by the session bean resumes.
4. `getBalance()` is called on the instance returned by `findByPrimaryKey`. The instance is hydrated and the access intent specified for this method is used. Any other method calls within the transaction will execute with the same access intent.

Note: WebSphere Application Server V6.0 and V6.1 also includes a feature (inherited from WebSphere Application Server V5 Enterprise Edition) that provides an extension to access intents called Application Profiles, which handles the problem mentioned above in a powerful way. Application profiles let you externally specify a set of tasks (that is, a flow of calls in your code), and specify which access intent should be used for a specific task. For information about Application Profiles, please refer to the WebSphere Information Center.

Choosing the right access intent

The main rule is: keep it simple.

Start with the default setting (wsPessimisticUpdate-WeakestLockAtLoad), and work from there. Specifying access intents on all your business methods could lead to a configuration, debugging, and maintenance nightmare. Specify access intents on a selected number of methods. Also, choose access intents wisely.

- ▶ Access intents can be applied to your business methods, to the `findByPrimaryKey()` method, as well as the create and remove method. As much as possible, avoid other methods.
- ▶ Make sure that no method is configured with more than one access intent policy. Applications that are misconfigured in this way will not be runnable until the configuration errors are fixed.
- ▶ For entity beans backed by tables with nullable columns, use optimistic policies with caution. Nullable columns are automatically excluded from overqualified updates at deployment time. This means that at commit time, those columns will not be used in the update statement to check whether the data has changed or not. Therefore, concurrent changes to a nullable field might result in lost updates. Using the Application Server Toolkit, you can set a property on each enterprise bean attribute called *OptimisticPredicate*, as shown in Figure 13-18 on page 868. You can change this property by editing the data mappings of your EJBs. When this property is set, the column, even if it is nullable, will be reflected in the overqualified update statement that is generated in the deployment code to support optimistic policies.

WebSphere Application Server supports an optimistic concurrency control scheme for EJB 2.x CMP entity beans that allows you to add a column for collision detection in your relational database table. This column is reserved to determine if a record has been updated. When using a collision detection column, the overqualified UPDATE statement only needs the collision detection column and the primary key. To manage the collision detection column, provide your own database trigger implementation. Using the collision detection column overcomes the nullable column limitation and the unsupported optimistic concurrency control data types, such as BLOBs and CLOBs.

Tip: If you want to check which SQL code is executed for an optimistic update, check the `storeUsingOCC` method in the `<bean>FunctionSet` generated class.

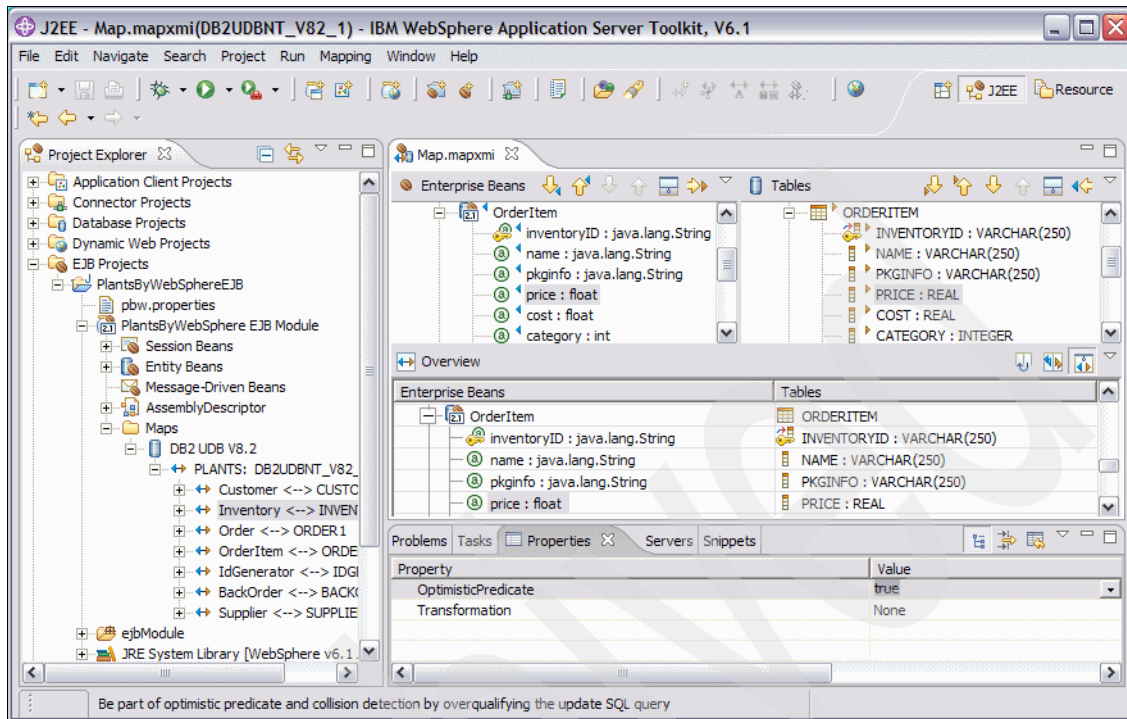


Figure 13-18 Optimistic predicate property

If a bean is loaded for read intent and an update is attempted during that transaction, then the persistence manager will raise an `UpdateCannotProceedWithIntegrity` exception. In other words, if you call `findByPrimaryKeyForRead()` and an update is attempted, it will fail.

Important: The behavior described above is true for all access intents except `wsPessimisticUpdate-NoCollision`. This access intent will not flag updates, even if no locks are held. Our recommendation is that you avoid this access intent in production.

13.5.4 Using read-ahead hints

Read-ahead schemes enable applications to minimize the number of database round trips by retrieving a working set of CMP beans for the transaction within one query. Read-ahead involves activating the requested CMP beans and caching the data for their related beans, which ensures that data is present for the beans that are most likely to be needed next by an application.

A read-ahead hint is a canonical representation of the related beans to be read. It is associated with a finder method for the requested bean type, which must be an EJB 2.x-compliant CMP entity bean. Currently, only `findByPrimaryKey` methods can have read-ahead hints. Only beans related to the requested beans by a container-managed relationship (CMR), either directly or indirectly through other beans, can be read ahead.

To set Read-ahead hints, do the following:

1. Open the EJB deployment descriptor editor.
2. Select the **Access** tab and scroll down to the WebSphere Extensions section.
3. Click the **Add** button to the right of the Access Intent for Entities 2.x (Method Level) field. See Figure 13-19.

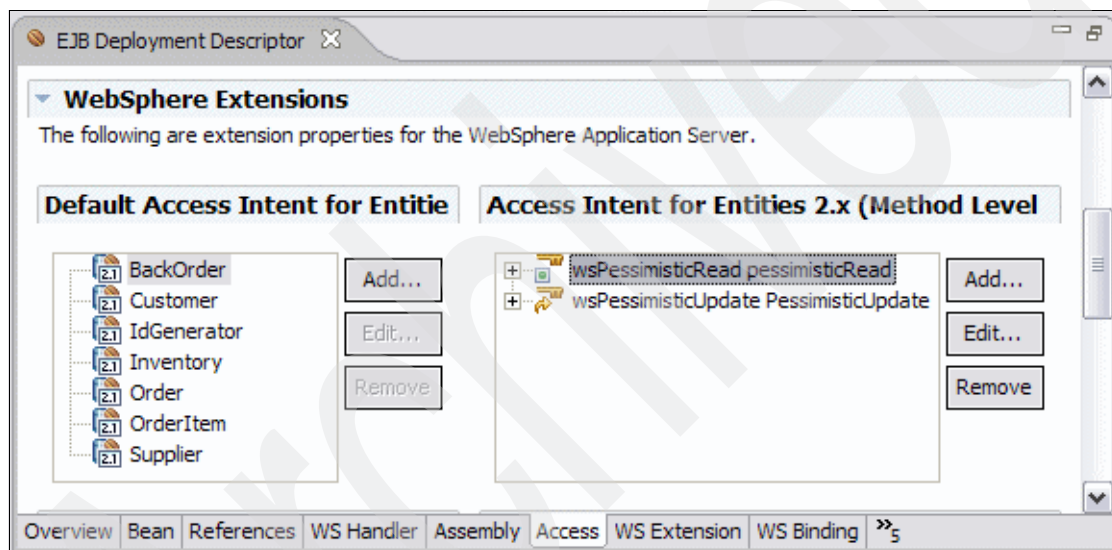


Figure 13-19 Adding a read ahead hint

See Figure 13-20 on page 870.

Edit 2.x Access Intent

Access Intent
Enter name and description.

Name: pessimisticRead

Access intent name: wsPessimisticRead

A value of read indicates the caller does not intend to drive an update method on the entity bean. This policy locks tables or rows being read in the SQL statement.

Description:

☒ Read ahead hint

Persistence option

☐ Verify read only data

☐ Deferred operation

☐ Batch

< Back Next > Finish Cancel

Figure 13-20 Specifying read-ahead hint

4. Follow the windows in the wizard to select the beans and methods, then click **Finish**.

Note: While using read-ahead hints can improve performance by minimizing the number of database round trips required to retrieve the data for the related beans, overdoing it can have an adverse effect. For example, using read-ahead hints on long or complex paths can result in a query that is too complex to be useful.

13.5.5 Tracing access intents behavior

You can obtain a very detailed trace of the EJB persistence manager behavior by specifying the following trace specification for an application server:

```
com.ibm.ejs.container.*=all=enabled:com.ibm.ejs.persistence.*=all=enabled:  
com.ibm.ws.appprofile.*=all=enabled.
```

13.6 IBM EJB extensions: inheritance relationships

Support for entity beans inheritance, which is not part of the EJB 1.1 nor EJB 2.x specifications, is also available in the toolkit. Support for enterprise entity beans relationships for EJB 1.1, although not standard, is also available using this tool. Refer to the toolkit documentation for more details.

13.7 IBM Web module extensions

WebSphere Application Server also provides multiple extensions for Web modules. To work with these extensions, open the Web deployment descriptor by double-clicking the Web module in the J2EE Hierarchy view. To see the IBM Web module extensions, select the **Extensions** tab, as in Figure 13-21.

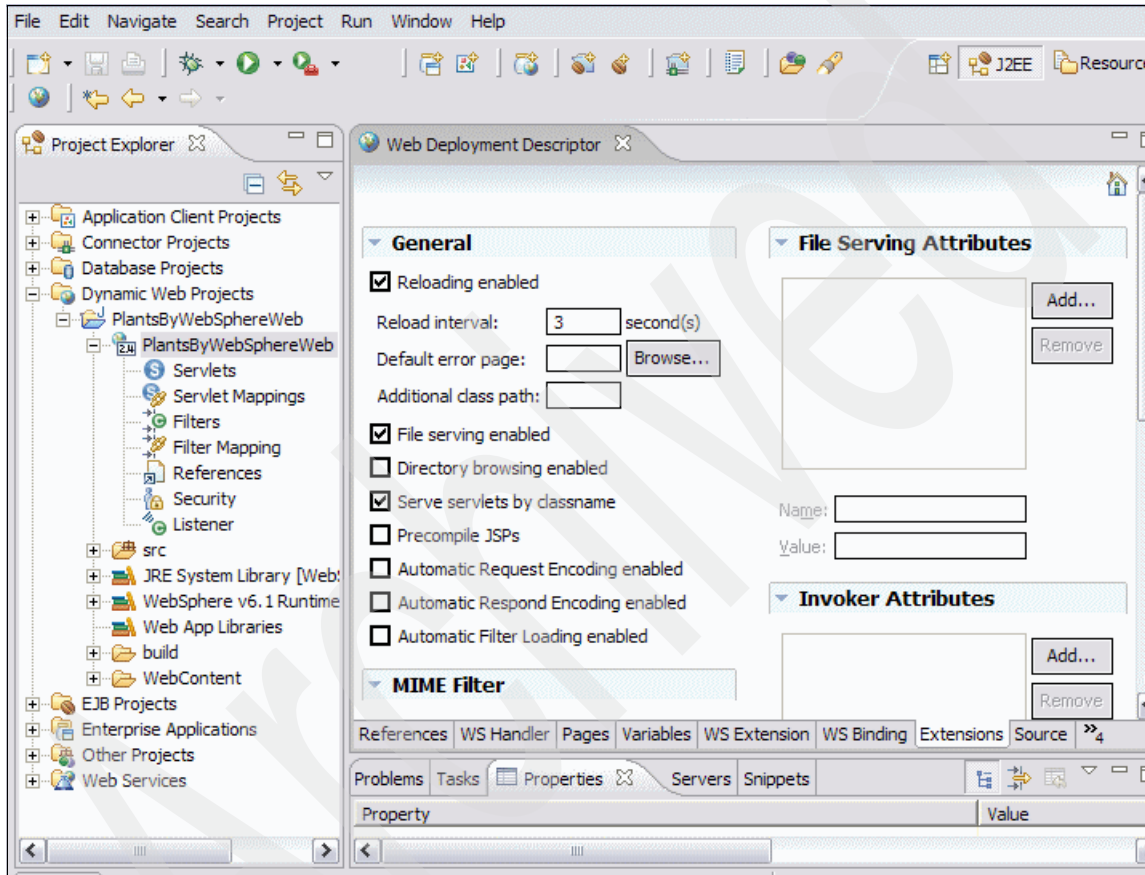


Figure 13-21 Web module extensions

13.7.1 File serving servlet

When dealing with static content (HTML pages, images, style sheets, and so on), you can choose to have these resources served by WebSphere, or have them served by the HTTP server itself.

If you want WebSphere to serve the static content of your application, you must enable file servlet, also known as the file serving servlet or file serving enabler.

This servlet serves up any resource file packaged in the WAR file. The File serving enabled attribute is set to true by default. By changing it to false, the Web server plug-in will not send requests for static content to WebSphere, but leave it up to the HTTP server to serve them.

If you want the Web server to serve static content, you can experience better performance than using WebSphere in this instance, because the Web server is serving the content directly. Moreover, a Web server has more customization options than the file servlet can offer.

However, using the WebSphere file serving servlet has the advantage of keeping the static content organized in a single, deployable unit with the rest of the application. Additionally, this allows you to protect the static pages using WebSphere security.

To enable this option, check the **File serving enabled** box. Enter attributes used by the file serving servlet in the File Serving Attributes section.

13.7.2 Web application auto reload

If you check the **Reloading enabled** option, the class path of the Web application is monitored and all components, JAR or class files, are reloaded whenever a component update is detected. The Web module's class loader is shut down and restarted. The reload interval is the interval between reloads of the Web application. It is set in seconds.

The auto reload feature plays a critical role in hot deployment and dynamic reload of your application.

Important: You must set the Reloading enabled option to true for JSP files to be reloaded when they are changed on the file system. Reloading a JSP does not trigger the reload of the Web module, because separate class loaders are used for servlets and JSP.

This option is set to true by default, with the reload interval set to three (3) seconds. In production mode, you might consider making the reload interval much higher.

13.7.3 Serve servlets by class name

The invoker servlet can be used to invoke servlets by class name. Note that there is a potential security risk with leaving this option set in production. It should be seen as more of a development-time feature, for quickly testing your servlets.

A better alternative than this option is to define servlet mappings in the Web deployment descriptor for the servlets that should be available.

This option is turned off by default.

13.7.4 Default error page

This page will be invoked to handle errors if no error page has been defined, or if none of the defined error pages matches the current error.

13.7.5 Directory browsing

This Boolean defines whether it is possible to browse the directory if no default page has been found.

This option is turned off by default.

13.7.6 JSP attributes

The JSP engines provides several options that can be set to customize its behavior:

- keepgenerated

If this Boolean is set to true, the source code of the servlet created by compilation of a JSP page is kept on the file system. Otherwise, it is deleted as soon as the servlet code has been compiled; only the .class file is available.

- scratchdir

This string represents the directory in which the generated class files will be generated. If this string is not set, code is created under:

`<profile_root>\temp`

- jdkSourceLevel

This setting specifies which JDK level JSPs will be generated and compiled for. Valid values are 13 (default), 14, and 15. To compile for Java 5.0, use 15.

There are several more options that can be set. Search for “JSP engine configuration parameters” in the WebSphere InfoCenter for the full list.

13.7.7 Automatic HTTP request and response encoding

The Web container no longer automatically sets request and response encodings and response content types. The programmer is expected to set these values using the methods available in the Servlet 2.4 API. If you want the application server to attempt to set these values automatically, check the **Automatic Request Encoding enabled** option in order to have the request encoding value set. Similarly, you can check the **Automatic Response Encoding enabled** option in order to have the response encoding and content type set.

The default value of the `autoRequestEncoding` and `autoResponseEncoding` extensions is false, which means that both the request and response character encoding is set to the Servlet 2.4 specification default of ISO-8859-1. Different character encodings are possible if the client defines character encoding in the request header, or if the code uses the `setCharacterEncoding(String encoding)` method.

If the `autoRequestEncoding` value is set to true, and the client did not specify character encoding in the request header, and the code does not include the `setCharacterEncoding(String encoding)` method, the Web container tries to determine the correct character encoding for the request parameters and data.

The Web container performs each step in the following list until a match is found:

- ▶ Looks at the character set (charset) in the Content-Type header.
- ▶ Attempts to map the server's locale to a character set using defined properties.
- ▶ Attempts to use the `DEFAULT_CLIENT_ENCODING` system property, if one is set.
- ▶ Uses the ISO-8859-1 character encoding as the default.

If you set the `autoResponseEncoding` value to true and the client:

- ▶ The client did not specify character encoding in the request header.
- ▶ The code does not include the `setCharacterEncoding(String encoding)` method,

The Web container does the following:

- ▶ Attempts to determine the response content type and character encoding from information in the request header.
- ▶ Uses the ISO-8859-1 character encoding as the default.

13.8 IBM EAR extensions: Sharing session context

In accordance with the servlet 2.4 API specification, the session manager supports session scoping by Web module only. Only servlets in the same Web module can access the data associated with a particular session. WebSphere Application Server provides an option that you can use to extend the scope of the session attributes to an enterprise application. Therefore, you can share session attributes across all the Web modules in an enterprise application.

This option can be set in the toolkit in the enterprise application deployment descriptor.

1. Open the deployment descriptor by double-clicking the enterprise application.
2. Check the **Shared session context** option in the WebSphere Extensions section. See Figure 13-22 on page 876.

The screenshot shows the 'EAR deployment descriptor' configuration window. It is divided into several sections:

- General Information:** Contains fields for 'Display name' (PlantsByWebSphere) and 'Description'.
- Security Roles:** Includes a 'Details...' button.
- Modules:** Lists 'Web PlantsByWebSphereWeb.war' and 'EJB PlantsByWebSphereEJB.jar', each with a 'Details...' button.
- Icons:** Contains 'Small' and 'Large' icon selection fields, each with a 'Browse...' button.
- WebSphere Extensions:** This section is expanded and contains:
 - A text label: 'The following are extension properties for the WebSphere Application Server.'
 - A 'Reload interval' field set to 'second(s)'.
 - A checked checkbox labeled 'Shared session context'.

Figure 13-22 EAR deployment descriptor

Important: To use this option, you must install all the Web modules in the enterprise application in the same application server. You cannot use this option when one Web module is installed in one server and the second Web module is installed in a different server.

In such split installations, applications might share session attributes across Web modules using distributed sessions. However, session data integrity is compromised when concurrent access to a session is made in different Web modules.

Sharing HTTP session context also severely restricts the use of some session management features, like time-based writes. For enterprise applications on which this option is enabled, the session management configuration set at the Web module level is ignored. Instead, the session management configuration defined at the enterprise application level is used.

13.9 Exporting the PlantsByWebSphere EAR file

Once you have made all the changes to your application and are ready to deploy, you need to export the EAR file to a location where it can be picked up for deployment by the application server.

To export the Plants by WebSphere sample application, do the following:

1. Select **File** → **Export**.
2. Select **EAR file** as the export target and click **Next**.
3. Select to export the **PlantsByWebSphere** EAR project and enter a suitable destination for the EAR file, such as C:\PlantsByWebSphere.ear. Then click **Finish**.

13.10 WebSphere Enhanced EAR

The Enhanced EAR, introduced in WebSphere Application Server V6.0, is a normal J2EE EAR file, but with additional configuration information for resources required by J2EE applications. While adding this extra configuration information at packaging time is not mandatory, it can simplify deployment of J2EE applications to WebSphere if the environments where the application is to be deployed is similar.

Table 13-6 shows the resources supported by the Enhanced EAR and the scope in which they are created.

Table 13-6 Scope for resources in WebSphere Enhanced EAR file

Resource	Scope
JDBC providers	Application
Data sources	Application
Resource adapters	Application
JMS resources	Application
Substitution variables	Application
Class loader policies	Application
Shared libraries	Server
JAAS authentication aliases	Cell
Virtual hosts	Cell

New in V6.1: The Enhanced EAR has been improved with V6.1 to include support for J2C Resource Adapters (RAR files) and JMS resources.

J2C Resource Adapters can be configured either as embedded or external resources. An embedded RAR is packaged within an enterprise application (EAR), deployed as a part of J2EE application installation, and is removed when the application is uninstalled from the server. An external RAR is packaged as a stand-alone RAR file, is deployed explicitly on a WebSphere node, and is not managed as a J2EE application. If an adapter is used only by a single application, it should be configured as an embedded RAR. If it is to be shared between multiple applications, it should be an external RAR.

When an Enhanced EAR is deployed to WebSphere Application Server, WebSphere can automatically configure the resources specified in the Enhanced EAR. This reduces the number of configuration steps required to set up the WebSphere environment to host the application.

When an Enhanced EAR is uninstalled, the resources that are defined at the application level scope are removed as well. However, resources defined at a scope other than application level are not removed because they might be in use by other applications.

Resources created at Application level scope are limited in visibility to only that application.

New in V6.1: In WebSphere Application Server V6.0, the administrative console did not allow you to view or change Application-scope settings. In V6.1, you can now do this. However, you cannot create new resources at the application scope using the administrative console. To add new application-scoped resources you must use the Application Server Toolkit or Rational Application Developer.

To view the application scoped resources, select **Applications** → **Enterprise Applications** → **<application>**. Select **Application scoped resources** in the References section. If there are no application scoped resources, you will not see this option.

13.10.1 Configuring a WebSphere Enhanced EAR

The supplemental information in an Enhanced EAR is modified by using the WebSphere Enhanced EAR editor, the Deployment tab of the application deployment descriptor in the Application Server Toolkit.

Note: Before adding or removing J2EE modules using the Module page in the Application Deployment Descriptor editor, do the following:

1. Click the **Deployment** tab to activate the functions in the deployment page.
2. Add your modules to the Module page.

Complete this task for each application deployment descriptor editor session that you want to add or remove modules from the Module page.

To access the Enhanced EAR deployment options, do the following:

1. In the J2EE Project Explorer view, expand **Enterprise Applications**, and then the application.

2. Double-click **Deployment Descriptor** and select the **Deployment** tab. This opens up the Enhanced EAR editor, as shown in Figure 13-23.

WebSphere Deployment

Data Sources
Allows the installed applications to access data from databases.

JDBC provider list:

Name	Implementation Class Name
Derby JDBC Provider (XA)	org.apache.derby.jdbc.EmbeddedXADataSource

Add Edit Remove

Data source defined in the JDBC provider selected above:

Name	JNDI Name	Type
DefaultEJBTimerDataSource	jdbc/DefaultEJBTimerDataSource	V5

Add Edit Remove

Resource properties defined in the data source selected above:

Name	Value	Type
databaseName	\${USER_INSTALL_ROOT}/databases/EJBTimers/\${SERVER}/EJBTimerDB	java.lang.String
shutdownDatabase		java.lang.String
dataSourceName		java.lang.String

Add Edit Remove

Embedded J2C Options

External J2C Options

Application

- Substitution Variables
- Authentication
- Shared Library
- Virtual Hosts

Overview Module Security **Deployment** Extended Services Source

Figure 13-23 WebSphere Enhanced EAR editor

In the Application section in Figure 13-24, you can see the class loader policies and class loader mode configured for each of the containing module. Plants by WebSphere runs fine with the default policies and modes, so they do not need to be changed.

Figure 13-24 Configuring class loader mode and class loader policies

In the Enhanced EAR editor, you also configure resources such as JDBC providers, datasources, JMS resources, class loader policies, JAAS authentication aliases, virtual hosts, and so forth.

To configure the Plants by WebSphere application, we need to add the following:

- ▶ JAAS authentication alias
- ▶ JDBC provider for DB2
- ▶ Data source for DB2 database

Just to show the editor, we will also configure a new virtual host for a domain called `www.plantsbywebsphere.com`.

Configuring a JAAS authentication alias

To configure the JAAS authentication alias, do the following:

1. In the Deployment tab, expand the **Authentication** section.
2. Click the **Add** button.
3. In the dialog box that displays, enter:
 - `plantsbywebsphere` as the alias
 - A user ID with access to the PLANTS database
 - The password for the user ID.
 - Plants by WebSphere as the description

4. Click **OK**. See Figure 13-25 on page 882.

Authentication
Allows to define a login configuration used by Java Authentication and Authorization Service (JAAS).

JAAS authentication list:

Alias	User ID	Description
plantsbywebsphere	db2admin	Plants by WebSphere

Add Edit Remove

Figure 13-25 Configuring JAAS authentication alias for Plants by WebSphere

Configuring a DB2 JDBC provider

To configure the DB2 JDBC provider, do the following:

1. Click the **Add** button next to the JDBC provider list in the **Data Sources** section.
2. In the dialog box:
 - Select **IBM DB2** as the Database type.
 - Select **DB2 Universal JDBC Driver Provider (XA)** as the JDBC provider type.

See Figure 13-26 on page 882.

Database type:

- User-defined
- IBM DB2**
- Cloudscape
- Informix
- Sybase
- Oracle

JDBC provider type:

- DB2 Universal JDBC Driver Provider
- DB2 Universal JDBC Driver Provider (XA)**
- DB2 Legacy CLI-based Type 2 JDBC Driver
- DB2 Legacy CLI-based Type 2 JDBC Driver (XA)
- DB2 UDB for iSeries (Native)
- DB2 UDB for iSeries (Native XA)

Description: XA DB2 Universal JDBC Driver-compliant Provider, Datasources created

Figure 13-26 Creating a DB2 JDBC Provider

Click **Next**.

3. In the next dialog box, enter a name for the JDBC provider (for administration purposes only) and leave the other properties as the default values. See Figure 13-27 on page 883.

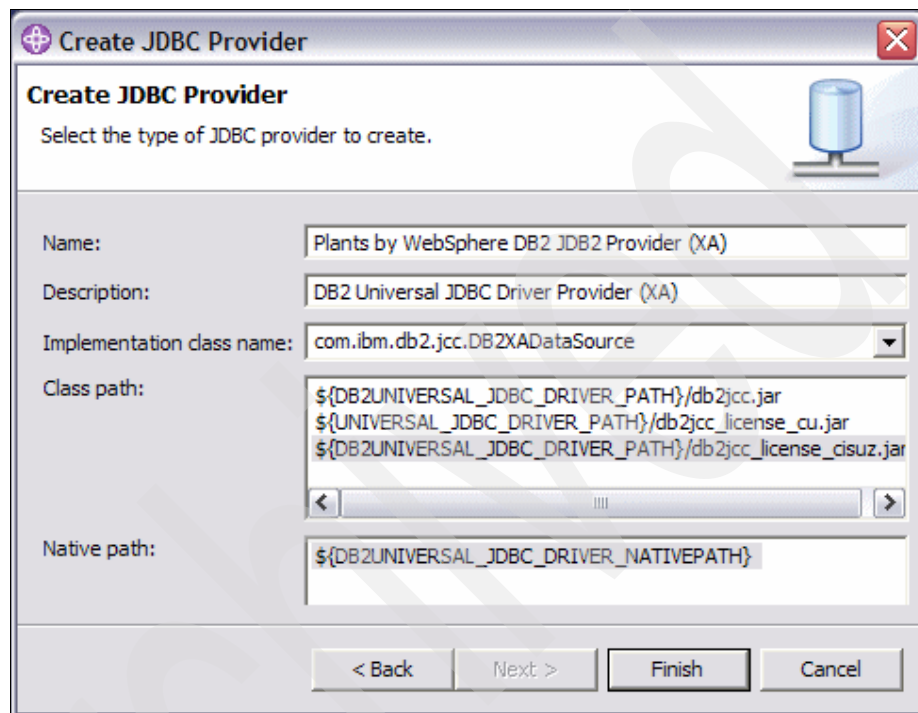


Figure 13-27 Creating a DB2 JDBC provider

Click **Finish**.

4. Select the **Plants by WebSphere DB2 JDBC Provider (XA)** you just created and click the **Add** button next to the Data source list, as in Figure 13-28.

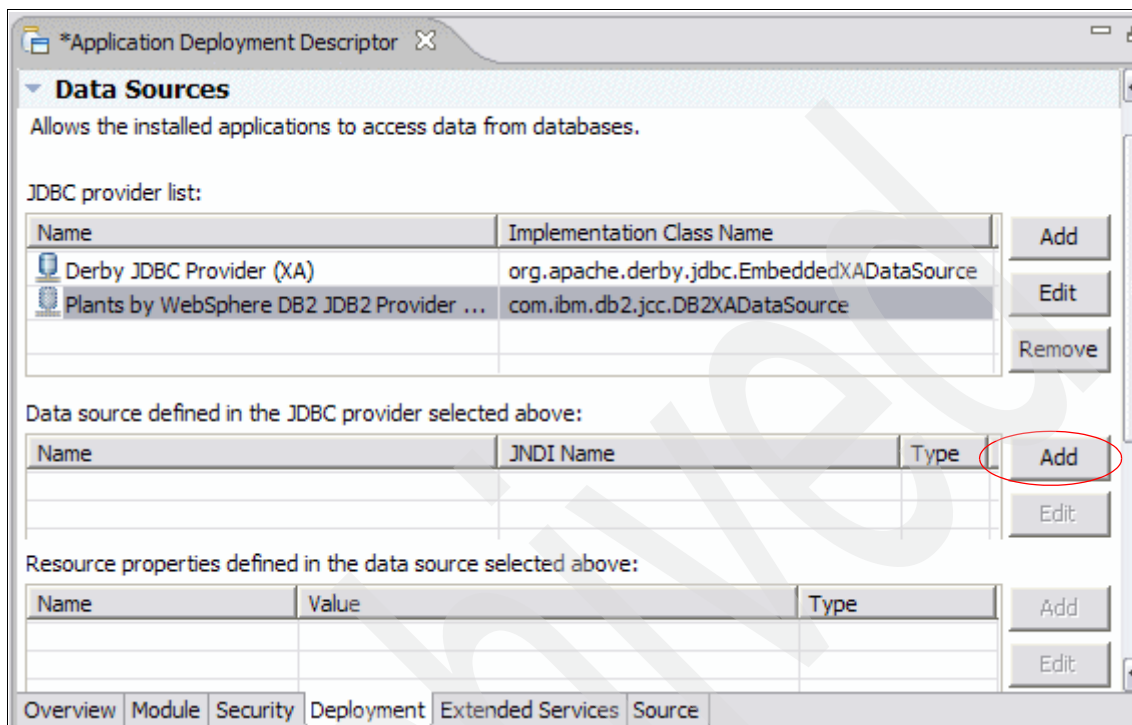


Figure 13-28 Add a data source

5. In the Create a Data Source dialog box, select **DB2 Universal JDBC Driver Provider (XA)** as the JDBC provider type and **Version 5.0 data source** as the data source type, as in Figure 13-29.

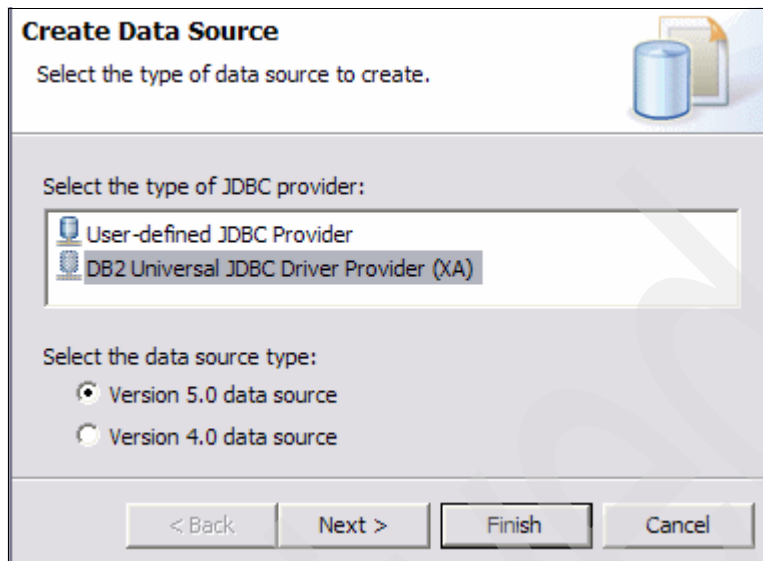
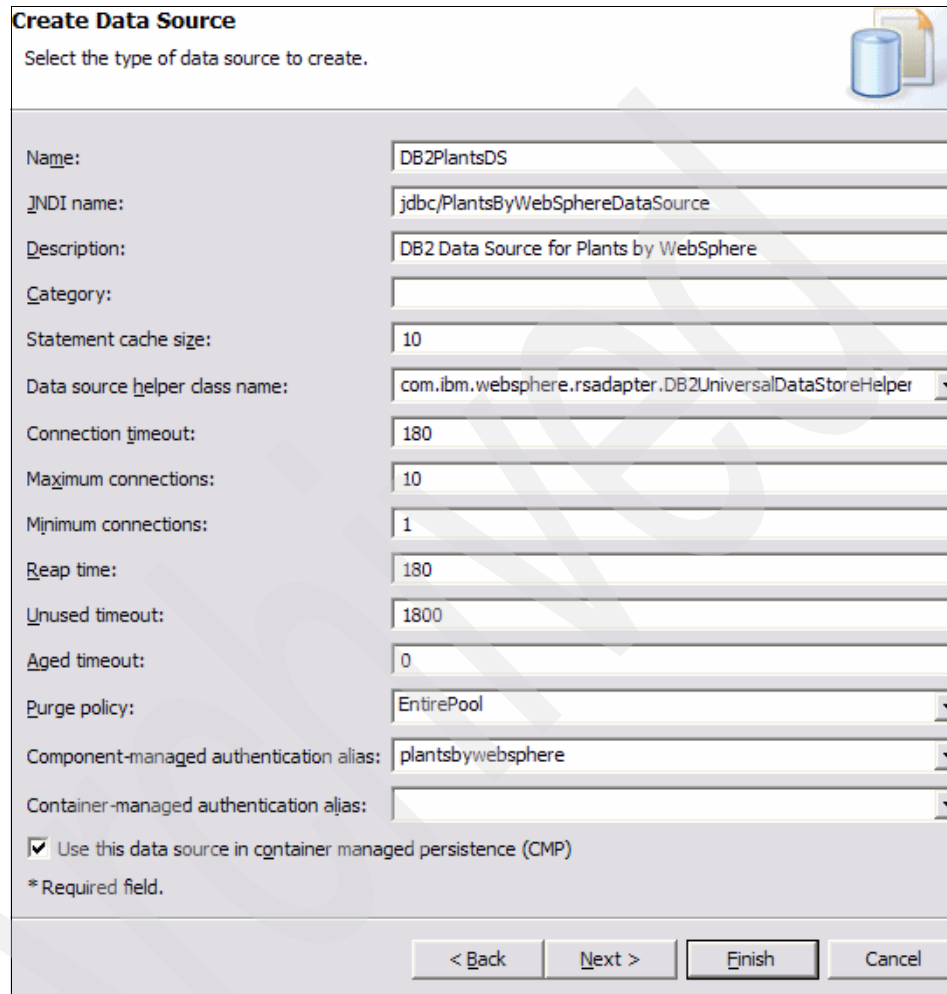


Figure 13-29 Creating a DB2 data source

Click **Next**.

6. In the dialog box displayed enter the appropriate values for the DB2 data source. See Figure 13-30 on page 886.



The image shows a 'Create Data Source' dialog box with a title bar and a close button. Below the title bar is a subtitle 'Select the type of data source to create.' and a small icon of a database cylinder. The main area contains several labeled text fields and dropdown menus. The fields are: Name (DB2PlantsDS), JNDI name (jdbc/PlantsByWebSphereDataSource), Description (DB2 Data Source for Plants by WebSphere), Category (empty), Statement cache size (10), Data source helper class name (com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper), Connection timeout (180), Maximum connections (10), Minimum connections (1), Reap time (180), Unused timeout (1800), Aged timeout (0), Purge policy (EntirePool), Component-managed authentication alias (plantsbywebsphere), and Container-managed authentication alias (empty). At the bottom, there is a checkbox labeled 'Use this data source in container managed persistence (CMP)' which is checked, and a note '* Required field.' Below the fields are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

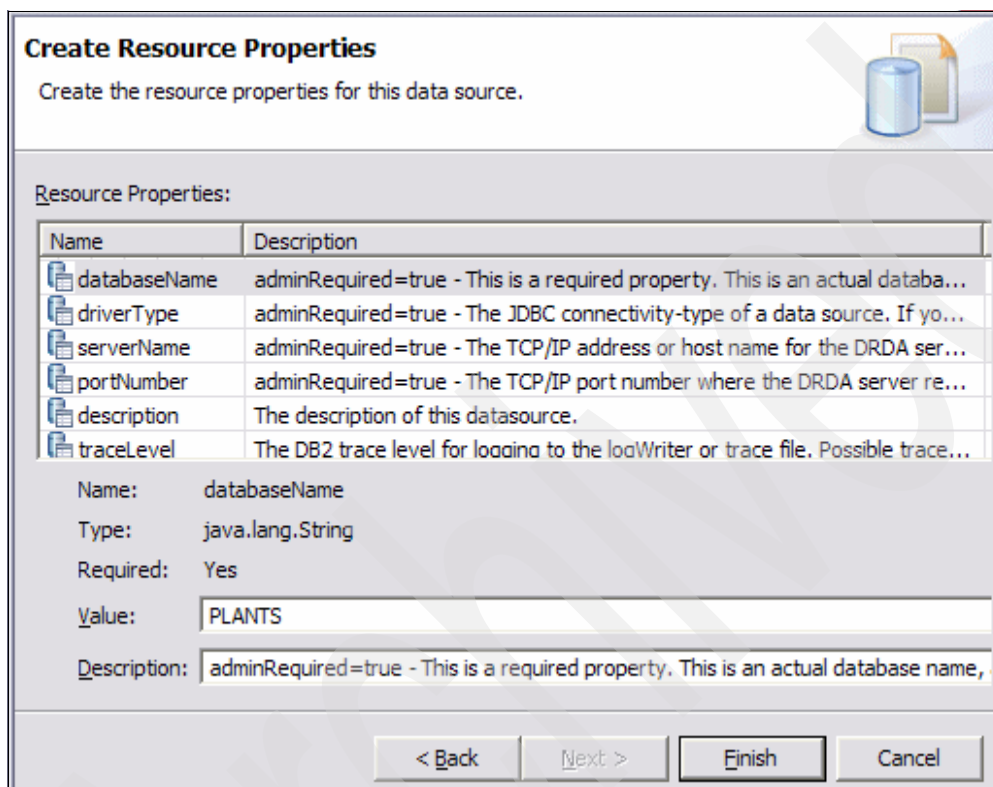
Name:	DB2PlantsDS
JNDI name:	jdbc/PlantsByWebSphereDataSource
Description:	DB2 Data Source for Plants by WebSphere
Category:	
Statement cache size:	10
Data source helper class name:	com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
Connection timeout:	180
Maximum connections:	10
Minimum connections:	1
Reap time:	180
Unused timeout:	1800
Aged timeout:	0
Purge policy:	EntirePool
Component-managed authentication alias:	plantsbywebsphere
Container-managed authentication alias:	
☒ Use this data source in container managed persistence (CMP)	
* Required field.	

Figure 13-30 Creating a DB2 data source

- Enter DB2PlantsDS as the name.
- Enter jdbc/PlantsByWebSphereDataSource as the JNDI name.
- Enter DB2 Data Source for Plants by WebSphere as the description.
- Select **plantsbywebsphere** as the Component-managed authentication alias.
- Check **Use this data source in container manager persistence (CMP)**.

Click **Next**.

7. In the Create Resource Properties dialog box, select **databaseName** and enter PLANTS as the value. Then also select the **driverType** and enter 2 as the value. See Figure 13-31 on page 887.



The dialog box is titled "Create Resource Properties" and contains the instruction "Create the resource properties for this data source." It features a table of resource properties and a section for editing a specific property.

Name	Description
databaseName	adminRequired=true - This is a required property. This is an actual databa...
driverType	adminRequired=true - The JDBC connectivity-type of a data source. If yo...
serverName	adminRequired=true - The TCP/IP address or host name for the DRDA ser...
portNumber	adminRequired=true - The TCP/IP port number where the DRDA server re...
description	The description of this datasource.
traceLevel	The DB2 trace level for loading to the logWriter or trace file. Possible trace...

Below the table, the "databaseName" property is selected for editing:

Name: databaseName
Type: java.lang.String
Required: Yes
Value: PLANTS
Description: adminRequired=true - This is a required property. This is an actual database name,

At the bottom are four buttons: "< Back", "Next >", "Finish", and "Cancel".

Figure 13-31 Setting database properties for DB2 data source

JDBC driver type 2 means that the database is local to the machine running the WebSphere application, or that it has a DB2 Connect™ client that can make the database look local.

Click **Finish**.

When you are finished, your data source configuration should look like Figure 13-32 on page 888.

WebSphere Deployment

Data Sources
Allows the installed applications to access data from databases.

JDBC provider list:

Name	Implementation Class Name
Derby JDBC Provider (XA)	org.apache.derby.jdbc.EmbeddedXADataSource
Plants by WebSphere DB2 JDB2 Provider ...	com.ibm.db2.jcc.DB2XADataSource

Data source defined in the JDBC provider selected above:

Name	JNDI Name	Type
DB2PlantsDS	jdbc/PlantsByWebSphereDataSource	V5

Resource properties defined in the data source selected above:

Name	Value	Type
databaseName	PLANTS	java.lang.String
driverType	2	java.lang.Integer
serverName		java.lang.String
portNumber	50000	java.lang.Integer

Overview | Module | Security | **Deployment** | Extended Services | Source

Figure 13-32 DB2 data source configured

Adding a virtual host

To configure a virtual host, do the following:

1. Expand the **Virtual Hosts** section of the Deployment tab and click the **Add** button next to the Virtual host name list.
2. In the Add Host Name Entry dialog box, enter `plantsbywebsphere_host` and click **OK**. Your new virtual host will appear in the Virtual Hosts list. See Figure 13-33.

Virtual Hosts
Enables a single host machine to resemble multiple host machines.

Virtual host name list:

Host Name	
plantsbywebsphere_host	

Add Edit Remove

Figure 13-33 Add a new virtual host

- Click the **Add** button next to the Host aliases list.
- In the Add Host Alias Entry dialog box, enter `www.plantsbywebsphere.com` for the host name and 80 for the port number. Click **OK**.

Repeat the procedure to add port number 9081 as well. We will use this port when we deploy the application later.

The host aliases will appear in the list (Figure 13-34).

Virtual Hosts
Enables a single host machine to resemble multiple host machines.

Virtual host name list:

Host Name	
plantsbywebsphere_host	

Add Edit Remove

Host aliases are defined by the virtual host name selected above:

Host Name	Port	
www.plantsbywebsphere.com	80	
www.plantsbywebsphere.com	9081	

Add Edit Remove

Figure 13-34 Configuring the virtual host for Plants by WebSphere

- When you are finished, press **Ctrl-S** to save the deployment descriptor editor.

Setting default virtual host for Web modules

Just because we have configured a new virtual host, `plantsbywebsphere_host`, in the Enhanced EAR file does not mean that all our Web modules automatically use it.

The default virtual host for a Web module created in the Application Server Toolkit or Rational Application Developer is `default_host`, which is also the case for the Web module of the Plants by WebSphere application. This setting can be found in the `ibm-web-bdn.xmi` file in the `/WEB-INF` directory of each Web module.

To configure the Web modules to default to the `plantsbywebsphere_host` instead, do the following:

1. Expand **Dynamic Web Project** in the Project Explorer view.
2. Expand the **PlantsByWebSphereWeb** project and double-click **Deployment Descriptor**.
3. Scroll to the bottom of the Overview page and replace `default_host` with `plantsbywebsphere_host`, as shown in Figure 13-35.

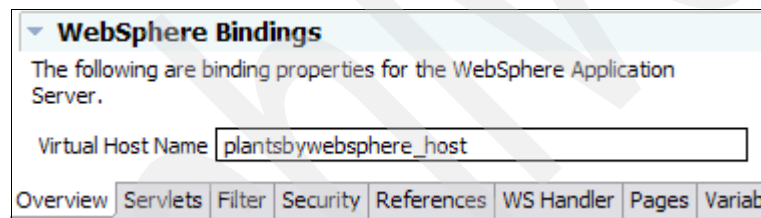


Figure 13-35 Setting default virtual host for a Web module

4. Save the deployment descriptor by pressing Ctrl-S and then close it.

Examining the WebSphere Enhanced EAR file

The information about the resources configured is stored in the `ibmconfig` subdirectory of the EAR file's `META-INF` directory. Expanding this directory reveals the well-known directory structure for a cell configuration, as seen in Figure 13-36 on page 891. You can also see the scope level where each resource is configured.

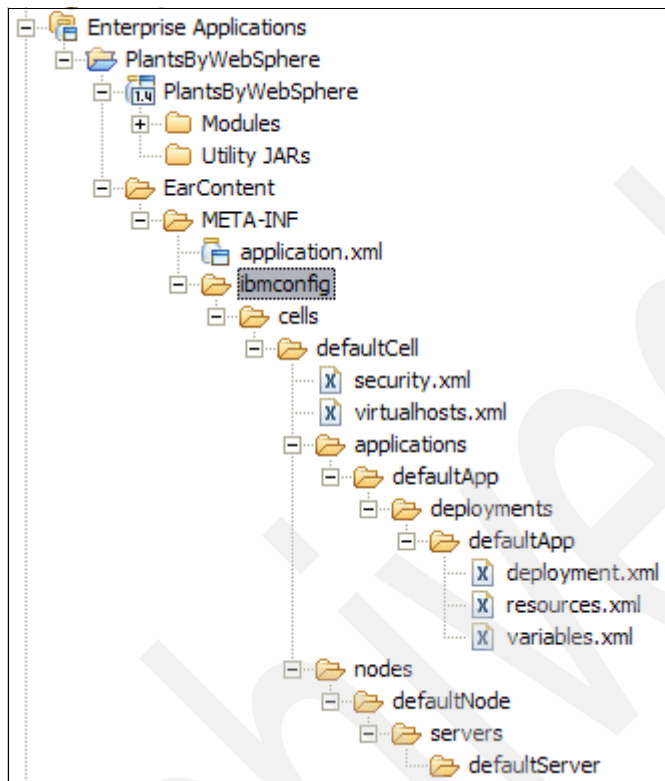


Figure 13-36 Enhanced EAR file contents

After you have re-packaged the application into an Enhanced EAR, export it as explained in “Exporting the PlantsByWebSphere EAR file” on page 877.

At deployment time, WebSphere Application Server uses this information to automatically create the resources.

13.11 Packaging recommendations

Here are some basic rules to consider when packaging an enterprise application:

- ▶ The EJB JAR modules and Web WAR modules comprising an application should be packaged together in the same EAR module.
- ▶ When a Web module accesses an EJB module, you should not package the EJB interfaces and stubs in the WAR modules. Thanks to the class loading architecture, EJB stubs and interfaces are visible by default to WAR modules.

- ▶ Utility classes used by a single Web module should be placed within its WEB-INF/lib folder.
- ▶ Utility classes used by multiple modules within an application should be placed at the root of the EAR file as Utility Projects, so they are accessible both by servlets and EJBs.
- ▶ Utility classes used by multiple applications can be placed on a directory referenced through a shared library definition.

See 12.5, “Learning class loaders by example” on page 811 for more details on how WebSphere finds and loads classes.

Deploying applications

In Chapter 13, “Packaging applications” on page 829, we discuss how to use the Application Server Toolkit to perform common tasks for packaging an application. In this chapter, we show you how to deploy the application. We take you through setting up the environment for the application, and then deploying the application itself. Next, we explain how to deploy the client part of the application. The deployment tasks in this chapter can also be automated using command-line tools, as explained in Chapter 5, “Administration with scripting” on page 249.

WebSphere Application Server V6 supports the J2EE Deployment API Specification (JSR-88), which defines standard APIs to enable deployment of J2EE applications and stand-alone modules to J2EE application servers. For more information about how to use this API, see the WebSphere Information Center by searching for JSR-88 and browse to the section discussing *Installing J2EE modules with JSR-88*.

The topics in this chapter include:

- ▶ Preparing the environment
- ▶ Generating deployment code
- ▶ Deploying the application
- ▶ Deploying application clients
- ▶ Updating applications

14.1 Preparing the environment

In this chapter, we show you how to set up a fairly complete environment for the Plants by WebSphere application and deploy the EAR file. You will not always need or want to customize the environment as extensively as we do in this chapter. Some steps are optional. If all you want to do is deploy your application quickly, using the WebSphere defaults for directory names, log files, and so forth, skip to 14.3, “Deploying the application” on page 913.

The steps in this section are performed typically by the application deployer. To deploy the Plants by WebSphere application, do the following:

1. Create the DB2 database for Plants by WebSphere. This step is required.
2. Create an environment variable for Plants by WebSphere server. This step is optional.
3. Create an application server to host the application. This step is optional.
4. Customize the IBM HTTP Server configuration. This step is optional.
5. Define a JDBC provider, data source, and authentication alias. This step is required if you are not using an Enhanced EAR.
6. Define virtual hosts. This step is optional and not required if you are using an Enhanced EAR.

If the application to be deployed is a WebSphere Enhanced EAR file, the resources configured in the Enhanced EAR file are created automatically when the application is deployed.

14.1.1 Creating the Plants by WebSphere DB2 database

The WebSphere samples by default use Cloudscape as the database. However, in this chapter, we will configure Plants by WebSphere to use a DB2 UDB 8.2 database instead.

To set up the DB2 database, make sure you have DB2 installed and running. Then run the following commands:

1. Select **Start → Programs → IBM DB2 → Command Line Tools → Command Window**.
2. Create the database using the commands in Example 14-1 on page 895.

Example 14-1 Creating the DB2 database

```
DB2 CREATE DATABASE PLANTS PAGESIZE 16 K
D2B CONNECT TO PLANTS USER <user_id> USING <password>
DB2 -tvf Table.ddl
db2 connect reset
```

The Table.ddl file is located in the PlantsByWebSphereEJB\ejbModule\META-INF\backends\DB2UDBNT_V82_1 directory of your Application Server Toolkit workspace. See “Creating a new database mapping and schema” on page 848.

Note: Because the Plants by WebSphere application stores images in the database using a BLOB, we needed a tablespace with a larger pagesize than the default of 4 KB. To specify this as the default when the database was created, we used the PAGESIZE keyword. This is supported on DB2 UDB 8.2.2, also called 8.1 Fix Pack 9, and later. We used DB2 Express 8.2.5

14.1.2 Creating an environment variable

We recommend that you use WebSphere environment variables, rather than hard-coded paths when deploying an application. In the following steps, we assume you have declared a PLANTSBYWEBSphere_ROOT variable. You will use it when specifying, for example, the JVM log's location.

Be certain you declare this variable at the right scope. For example, if you define this variable at the application server scope, it will only be known at that level. As long as you work with the WebSphere Application Server Base or Express editions, this is fine. But if you later decide to use the Network Deployment edition and you create a cluster of application servers, the PLANTSBYWEBSphere_ROOT variable will need to be defined at the cluster or cell level.

Use the steps in 4.1.10, “Using variables” on page 156 to create a PLANTSBYWEBSphere_ROOT variable with a value of C:\apps\PlantsByWebSphere.

There are several ways to organize WebSphere applications. Some companies prefer to create a directory for each application, as we do in our example, such as C:\apps\<application_name>, and keep all resources and directories required by the application in subdirectories under this directory. This strategy works well when deploying only one application per application server, again as we do in our example, because the application server's log files could then all be changed to point to c:\apps\<application_name>\logs.

Other companies prefer to organize resources by resource type, and so create directories such as `c:\apps\logs\<application_name.log>`, `c:\apps\properties\<application_name.properties>`, and so on.

And some companies prefer to stick with the vendor defaults as far as possible. For WebSphere, that means that the applications are installed in the `<profile_home>/installedApps` directory and the logs files are written to the `<profile_home>/logs/<server_name>` directory.

Which option you choose is a matter of personal preferences and corporate guidelines.

Note: Make sure you create the target directory you specify for the `PLANTSBYWEBSPPHERE_ROOT` variable before proceeding. If the directory is not created, the application server will not start.

14.1.3 Creating the Plants by WebSphere application server

In a distributed server environment, you have the option of using a single application server, or creating multiple application servers or clusters.

The advantages of deploying multiple applications to a single application server is that it consumes less resources. There is no overhead for any extra application server processes. Another benefit is that applications can make in-process calls to each other. For example, servlets in one EAR file could access Local interfaces of EJBs in another EAR file.

One alternative to using a single application server is to deploy each application to its own server. The advantages of deploying only one application on an application server is that it gives you greater control over the environment. The JVM heap sizes and environment variables are set at application server level, so all applications running in an application server share the JVM memory given to the application server and they would all see the same environment variables. Running each application in its own application server could also make it easier to perform problem determination. For example, if an application runs amok and consumes a lot of CPU, you could see which application it is by looking at the process ID of the application server.

In our example, we create a unique application server on which to run the Plants by WebSphere sample application.

Note: For a full discussion of application server properties, see 4.4, “Working with application servers” on page 170.

To create an application server, do the following:

1. Select **Servers** → **Application Servers**.
2. Click the **New** button and provide the information shown in Figure 14-1 on page 897.

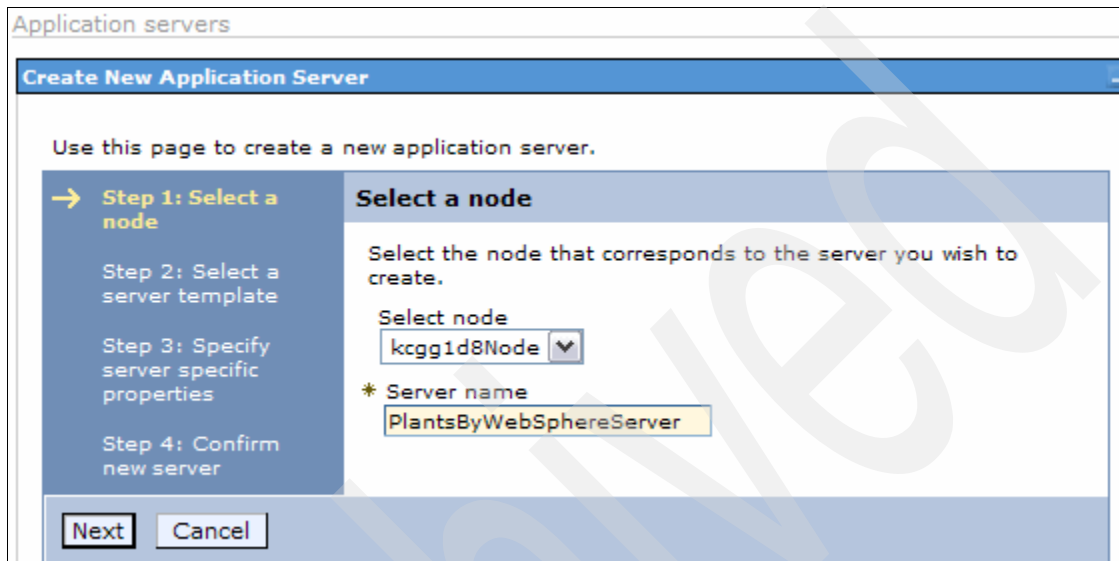


Figure 14-1 Creating the WebSphere Bank application server

- Node
Select the node on which the application server will be created.
 - Server name
Enter the application server name, such as PlantsByWebSphereServer.
- Click **Next**.
3. In Step 2, select which server template to use as the base for this new application server. The DeveloperServer template is used when setting up a server for development use and will cause the JVM to prioritize quick startup (by disabling bytecode verification, and performing JIT compilations with a lower optimization level). This option should not be used on a production server, where long run throughput is more important than early server startup. If you have not created any templates on your own, then select the WebSphere **default**. Otherwise, select the server template you want to use and click **Next**.
 4. In step 3, you can select, if you want, WebSphere to generate a unique set of port numbers for this application server. This ensures the ports defined for

this server does not conflict with another server currently configured on this node. Check the **Generate Unique Http Ports** box and click **Next**.

5. On the Summary page, click **Finish**.

Changing the working directory

The next thing we want to do is to change the working directory for the application server process. This directory is the relative root for searching files. For example, if you do a `File.open("foo.gif")`, `foo.gif` must be present in the working directory. This directory will be created by WebSphere if it does not exist. We recommend that you create a specific working directory for each application server.

1. Select the server, **PlantsByWebSphereServer**, you just created.
2. Expand the **Java and Process Management** in the Server Infrastructure section and select **Process Definition**.
3. Scroll down the page and change the working directory from `${USER_INSTALL_ROOT}` to `${PLANTSBYWEBSPPHERE_ROOT}/workingDir`.
4. Click **OK**.

Note: The working directory will not be created if you use a composed path, such as `C:/apps/PlantsByWebSphere/workingDir`. If you want to use such a path, create it before starting the application server, or the startup sequence fails.

Changing the logging and tracing options

Next, we want to customize the logging and tracing properties for the new application server. There are several ways to access the logging and tracing properties for an application server:

- ▶ Select **Troubleshooting** → **Logs and Trace** in the navigation bar, then select a server.
- ▶ Select **Servers** → **Application Servers**, select a server, and then select **Logging and Tracing** from the Troubleshooting section.
- ▶ Select **Servers** → **Application Servers**, select a server, select **Process definition** from the Java and Process Management section. Select **Logging and Tracing** from the Additional Properties section.

Because we have just finished updating the application server process definition, we will take the third navigation path to customize the location of the JVM logs, the diagnostic trace logs, and the process logs.

1. Select **Logging and Tracing**.

2. Select **JVM Logs**.

This allows you to change the JVM standard output and error file properties. Both are rotating files. You can choose to save the current file and create a new one, either when it reaches a certain size, or at a specific moment during the day. You can also choose to disable the output of calls to `System.out.print()` or `System.err.print()`.

We recommend that you specify a new file name, using an environment variable to specify it, such as:

```
${PLANTSBYWEBSphere_ROOT}/logs/SystemOut.log  
${PLANTSBYWEBSphere_ROOT}/logs/SystemErr.log
```

Click **OK**.

3. Select **Diagnostic Trace**.

Each component of the WebSphere Application Server is enabled for tracing with the JRes interface. This trace can be changed dynamically while the process is running using the Runtime tab, or added to the application server definition from the Configuration tab. As shown in Figure 14-2, the trace output can be either directed to memory or to a rotating trace file.

Change the trace output file name so the trace is stored in a specific location for the server using the `PLANTSBYWEBSphere_ROOT` variable and select the **Log Analyzer** format.

Configuration

General Properties **Additional Properties**

☒ Enable Log ☐ [Change Log Detail Levels](#)

Trace Output

☐ Memory Buffer

* Maximum Buffer Size
8 thousand entries

☒ File

* Maximum File Size
20 MB

* Maximum Number of Historical Files
1

* File Name
\${PLANTSBYWEBSphere_ROOT}/trace.log

Trace Output Format
Log Analyzer

Apply OK Reset Cancel

Figure 14-2 Specifying diagnostic trace service options

Click **OK**.

4. Select **Process Logs**.

Messages written by native code (JNI) to standard out and standard error streams are redirected by WebSphere to process logs, usually called native_stdout.log and native_stderr.log. Change the native process logs to:

`${PLANTSBYWEBSphere_ROOT}/logs/native_stdout.log`
`${PLANTSBYWEBSphere_ROOT}/logs/native_stderr.log`

Click **OK**.

5. All log files produced by the application server are now redirected to the `${PLANTSBYWEBSphere_ROOT}/logs` directory. Save the configuration.

Note: The rest of this example assumes a default HTTP port of 9081 for the Web container. Before proceeding, check the application server you created to determine the port you should use:

1. Select **Servers** → **Application Servers**.
2. Select the **PlantsByWebSphereServer**.
3. Select **Ports** in the Communications section.
4. Scroll down the page and note the port listed for WC_defaulthost.

14.1.4 Defining the Plants by WebSphere virtual host

Enhanced EAR file users: If you are using an Enhanced EAR file, the virtual host can be defined at packaging time. See “Adding a virtual host” on page 888.

Web modules need to be bound to a specific virtual host. For our sample, we chose to bind the PlantsByWebSphereWeb module to a specific virtual host called plantsbywebsphere_host. This virtual host has the following host aliases:

- ▶ www.plantsbywebsphere.com:80
- ▶ www.plantsbywebsphere.com:9081

Any request starting with <plantsbywebsphere_host_alias>/PlantsByWebSphere, such as `http://www.plantsbywebsphere.com:9081/PlantsByWebSphere`, is served by the Plants by WebSphere application.

Tip: You can restrict the list of hosts used to access the Web application by removing hosts from the virtual host definition.

Imagine you want to prevent users from directly accessing the Plants by WebSphere application from the WebSphere internal HTTP server when they invoke `http://www.plantsbywebsphere.com:9081/PlantsByWebSphere`. In other words, you want to force all requests to go through the Web server plug-in. You can achieve this by removing `www.plantsbywebsphere.com:9081` from the virtual host aliases list.

To create the plantsbywebsphere_host virtual host, do the following:

1. Select the **Environment** → **Virtual Hosts** entry in the navigation pane.
2. Click **New**.
3. Enter the virtual host name, plantsbywebsphere_host.
4. Click **Apply**.
5. Select **Host Aliases** in the Additional Properties section.

6. Add the two aliases shown in Figure 14-3 by clicking **New**, entering the values, and clicking **OK**.

New Delete    		
Select	Host Name ↕	Port ↕
☐	www.plantsbywebsphere.com	80
☐	www.plantsbywebsphere.com	9081
Total 2		

Figure 14-3 WebSphere Bank virtual host aliases

7. Click **OK**.
8. Save the configuration.

14.1.5 Creating the virtual host for IBM HTTP Server and Apache

Now that we have defined a `plantsbywebsphere_host` virtual host, we need to configure the Web server to serve the host aliases in the virtual host. The steps below are valid for both the IBM HTTP Server V6 and Apache 2.0.

Configuring virtual hosting

Note: It is not necessary to create a virtual host in `httpd.conf`. It is required only if you want to customize the configuration, for example, by separating the logs for each virtual host. This is not normally done.

Creating virtual hosts is done using the `VirtualHost` directive, as in Example 14-2.

Example 14-2 Using `VirtualHost`

```
<VirtualHost www.plantsbywebsphere.com:80>
  ServerAdmin webmaster@plantsbywebsphere.com
  ServerName www.plantsbywebsphere.com
  DocumentRoot "C:\IBM\HTTPServer\htdocs\plantsbywebsphere"
  ErrorLog logs/plantsbywebsphere_error.log
  TransferLog logs/plantsbywebsphere_access.log
</VirtualHost>
```

If you want to have multiple virtual hosts for the same IP address, you must use the `NameVirtualHost` directive. See Example 14-3.

Example 14-3 Using the `NameVirtualHost` and `VirtualHost` directives

NameVirtualHost 9.23.456.789:80

```
<VirtualHost itso_server:80>
  ServerAdmin webmaster@itso_server.com
  ServerName itso_server
  DocumentRoot "C:\IBM\HTTPServer\htdocs\itso_server"
  ErrorLog logs/itso_server_error.log
  TransferLog logs/itso_server_access.log
</VirtualHost>

<VirtualHost www.plantsbywebsphere.com:80>
  ServerAdmin webmaster@plantsbywebsphere.com
  ServerName www.plantsbywebsphere.com
  DocumentRoot "C:\IBM\HTTPServer\htdocs\plantsbywebsphere"
  ErrorLog logs/plantsbywebsphere_error.log
  TransferLog logs/plantsbywebsphere_access.log
</VirtualHost>
```

The `www.plantsbywebsphere.com` and the `itso_server` hosts have the same IP address, 9.23.456.789. We have set this by inserting the following line in the machine hosts file, located in `%windir%\system32\drivers\etc` or in `/etc` on UNIX systems):

```
9.23.456.789 www.plantsbywebsphere.com itso_server
```

In a real-life environment, this would probably be achieved by creating aliases at the DNS level. In any event, you must be able to ping the host you have defined, using commands such as **ping `www.plantsbywebsphere.com`**.

As you can see in Example 14-3, each virtual host has a different document root. Make sure that the directory you specify exists before you start the HTTP server. While testing the setup, you can place an `index.html` file at the document root stating which virtual host is being called. This lets you easily see which virtual host is being used.

You must restart the IBM HTTP Server to apply these changes. If you are running a Windows system, we recommend that you try to start the server by running **apache.exe** from the command line rather than from the Services window. This allows you to spot error messages thrown at server startup.

If your virtual hosts are correctly configured, invoking `http://www.plantsbywebsphere.com` or `http://itso_server` returns different HTML pages.

14.1.6 Creating a DB2 JDBC provider and data source

Enhanced EAR file users: If you are using an Enhanced EAR file, the JDBC provider, data source, and J2C authentication entry can be defined at packaging time. See “Configuring a DB2 JDBC provider” on page 882.

The Plants by WebSphere sample application uses a relational database, via entity beans, to store information. To access this database, you need to define a data source and then associate it with the entity beans. The Plants by WebSphere sample application is configured for Cloudscape by default. In Chapter 13, “Packaging applications” on page 829, however, we modified the Plants by WebSphere application to run against a DB2 database instead. We will now create the DB2 JDBC provider, data source, and JAAS authentication alias required to run against DB2.

For detailed information about JDBC providers and data sources, refer to 6.2, “JDBC resources” on page 305.

Configuring environment variables for DB2 JDBC driver

For the DB2 Universal JDBC Provider to find its classes, the DB2UNIVERSAL_JDBC_DRIVER_PATH and DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH environment variables must be set up. To set up these variables, do the following:

1. Select **Environment** → **WebSphere Variables**.
2. Locate and click the **DB2UNIVERSAL_JDBC_DRIVER_PATH** entry.
3. In the value field, enter the path to where the DB2 JDBC driver is located. For example, for DB2, the location is likely to be:

C:\Program Files\IBM\SQLLIB\java

See Figure 14-4.

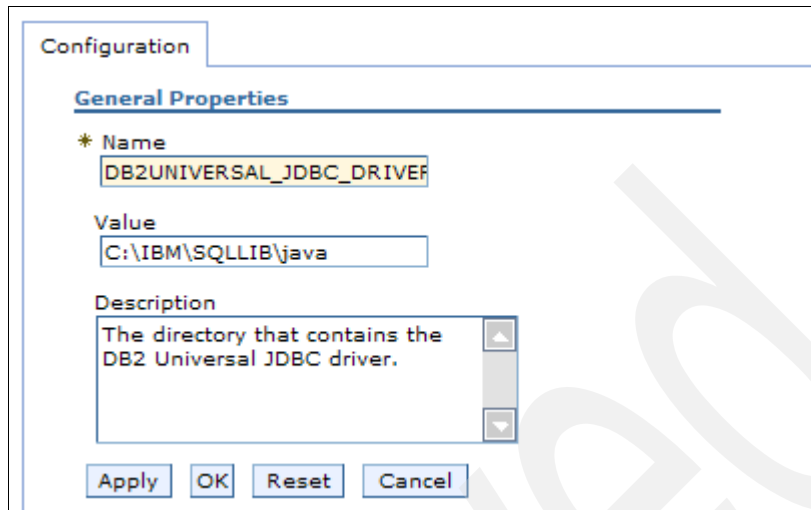


Figure 14-4 Configuring DB2 Driver Path

Click **OK**.

4. Repeat the process for the DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH variable. For DB2, it should use the same path, C:\Program Files\IBM\SQLLIB\java.

Configuring J2C authentication data

The user ID and password required to access the database are specified in a J2C authentication data entry.

1. Select **Security** → **Secure administration, applications, and infrastructure**. Expand the Java Authentication and Authorization Service section and select **J2C authentication data**.

2. Click **New**, and specify the following information to create the authentication data. Once completed, the authentication information should be similar to Figure 14-5.
 - Alias
Enter the name of the security information alias, such as webspherebank.
 - User ID
Enter a user ID with the proper authority to access the database.
 - Password
Enter the password for the user ID.

Secure administration, applications, and infrastructure

Secure administration, applications, and infrastructure > JAAS - J2C authentication data > New

Specifies a list of user identities and passwords for Java(TM) 2 connector security to use.

Configuration

General Properties

* Alias
plantsbywebsphere

* User ID
db2admin

* Password
.....

Description
Plants by WebSphere authen

Apply OK Reset Cancel

Figure 14-5 Creating WebSphere Bank JAAS authentication alias

3. Click **OK**.

Creating the Plants by WebSphere JDBC provider

The following steps take you through the creation of a JDBC provider targeting a DB2 database. To create a JDBC provider from the administrative console, do the following:

1. Expand the **Resources** entry and select the **JDBC Providers** entry.

2. Select the scope of this resource. In a stand-alone server environment, it is sufficient to create the data source at the server level. Otherwise, define it at the cluster or cell level. A rationale for this is to be able to share the definition across multiple servers in a cluster. To change this, select the server you are deploying to in the scopes list and click **Apply**.
3. Click the **New** button.
4. In the Configuration dialog box, select the general properties for the JDBC provider, as shown in Figure 14-6.

→ Step 1: Create new JDBC provider

Step 2: Enter database class path information

Step 3: Summary

Create new JDBC provider

Set the basic configuration values of a JDBC provider, which encapsulates the specific vendor JDBC driver implementation classes that are required to access the database. The wizard fills in the name and the description fields, but you can type different values.

Scope
cells:kcg1d8Cell:nodes:kcg1d8Node

* Database type
DB2

* Provider type
DB2 Universal JDBC Driver Provider

* Implementation type
XA data source

* Name
DB2 Universal JDBC Driver Provider (XA)

Description
XA DB2 Universal JDBC Driver-compliant Provider. Datasources created under this provider support the use of XA to perform 2-phase commit processing. Use of driver type 2 on WAS z/OS is not supported for datasources created under this provider.

Next Cancel

Figure 14-6 Creating a DB2 JDBC provider

- Database type: DB2
- Provider type: DB2 Universal JDBC Driver Provider
- Implementation type: XA data source

- Name: DB2 Universal JDBC Driver Provider (XA)

Note: We used the DB2 XA-capable JDBC Driver for the Plants by WebSphere sample. If your application does not require two-phase commit capabilities, use the regular driver. If using an XA-capable driver, it is a best practice to indicate that it is an XA-capable driver by including XA in its name, such as MyJDBCdriverXA.

5. The next window allows you to change the location for the JDBC driver files, but because we configured the paths earlier, we do not need to do it again. Click **Next**.
6. On the Summary page, click **Finish**.

Creating the Plants by WebSphere data source

The next step is to create the data source for the Plants by WebSphere DB2 database. To create a data source, do the following:

1. Select **Resources** → **JDBC Providers**.
2. Select the **DB2 Universal JDBC Driver Provider (XA)** and select **Data Sources** under Additional Properties.
3. Click **New** to add the new data source. See Figure 14-7.

Create a data source

→ Step 1: Enter basic data source information

Step 2: Enter database specific properties for the data source

Step 3: Summary

Enter basic data source information

Set the basic configuration values of a data source for association with your JDBC provider. A data source supplies the physical connections between your application server and the database.

Requirement: Use the Data sources (WebSphere(R) Application Server) console pages if your applications are based on the Enterprise JavaBeans (TM) (EJB) 1.0 specification or the Java(TM) Servlet 2.2 specification.

Scope

cells:kcgg1d8Cell:nodes:kcgg1d8Node

JDBC provider name

DB2 Universal JDBC Driver Provider (XA)

* Data source name

PlantsByWebSphereDS

* JNDI name

jdbc/PlantsByWebSphereDataSource

Component-managed authentication alias and XA recovery authentication alias

Select a component-managed authentication alias. The selected authentication alias will also be set as the XA recovery authentication alias if your JDBC Provider supports XA. If you choose to [create a new J2C authentication alias](#), the wizard will be canceled.

kcgg1d8CellManager/plantsbywebsphere ▼

Next

Cancel

Figure 14-7 Plants by WebSphere basic data source properties

- Data source name
Enter the data source name, which must be unique in the administrative domain or cell. We recommend that you use a value indicating the name of the database this data source is targeting, such as “PlantsByWebSphereDS”.
- JNDI name
Enter the name by which applications access this data source. If not specified, the JNDI name defaults to the data source name prefixed with jdbc/. For the Plants by WebSphere, set this field to jdbc/PlantsByWebSphereDataSource. This value can be changed at any time after the data source has been created.

- Component-managed authentication alias

Enter the J2C alias used for connecting to the data source by selecting the authentication alias created previously, <cell name>/plantsbywebsphere.

Note: The component-managed authentication alias is used when the res-auth tag in the deployment descriptor is set to Application and the application itself does not specify a user ID when obtaining a connection (that is, when `datasource.getConnection()` is called). The container-managed application alias (used when res-auth tag is set to Container) has been deprecated since WebSphere Application Server V6.0.

4. Click **Next**. On the second page, enter the information shown in Figure 14-8.

Step 1: Enter basic data source information

→ **Step 2: Enter database specific properties for the data source**

Step 3: Summary

Enter database specific properties for the data source

Set these database-specific properties, which are required by the vendor JDBC driver to support the connections that are made to this data source.

* Database name
PLANTS

* Driver type
2

Server name

Port number
50000

☒ Use this data source in container managed persistence

Previous Next Cancel

Figure 14-8 Plants by WebSphere database specific properties

- Database name

Enter the name of the database, PLANTS in our example.

- Driver type

Select the driver type to use. In our environment, the database is on the same machine as our WebSphere installation, so we can use a type 2 driver and do not need to enter the server name. If the database is on a remote server from the WebSphere machine, you would either use a type

- 4 driver (and supply the name of the database server) or you would use a type 2 driver and catalog the database on the WebSphere machine.
- Use this Data Source in container-managed persistence (CMP)
 - Check this option to indicate that the Plants by WebSphere data source is used for persisting the application entity beans.
- 5. Click **Next**, and then on the summary page, click **OK**.
- 6. Save the configuration.
- 7. Test the connection by selecting the data source and clicking the **Test Connection** button.

14.2 Generating deployment code

At some point, you will need to generate the deployment code for the EJBs. You can do this in Rational Application Developer, in the Application Server Toolkit, from the command line, or at deployment time using the install windows in the WebSphere administrative console. The deployment code should match the version of the run time target.

For information about how to generate the deployment code using the Application Server Toolkit, see “Creating a new database mapping and schema” on page 848.

14.2.1 Using EJBDeploy command-line tool

You can generate the EJB deployment code using the EJBDeploy command-line tool. The syntax of the EJBDeploy command is shown in Example 14-4.

Example 14-4 EJBDeploy syntax

EJBDeploy (v6.1, o0619.34)

Syntax: EJBDeploy inputEar workingDirectory outputEar [options]

Options:

-cp "jar1;jar2"	List of jar filenames required on classpath
-codegen	Only generate the deployment code, do not run RMIC or Javac
-bindear:options	Bind references within the EAR
-dbschema schema	The name of the schema to create
-dbvendor DBTYPE	Set the database vendor type, to one of:
DB2UDB_V81 DB2UDB_V82	
DB2UDBOS390_V7 DB2UDBOS390_V8 DB2UDBOS390_NEWFN_V8	
DB2UDBISERIES_V53 DB2UDBISERIES_V54	
DERBY_V10	
INFORMIX_V93	INFORMIX_V94 INFORMIX_V100

MSSQLSERVER_2000	MSSQLSERVER_2005
ORACLE_V9I	ORACLE_V10G
SYBASE_V1250	SYBASE_V15
SQL92 (*)	SQL99 (*) *Deprecated
-debug	Compile the code with java debug information
-keep	Do not delete the contents of the working directory
-ignoreErrors	Do not halt for compilation or validation errors
-quiet	Only display errors, suppress informational messages
-nowarn	Disable warning and informational messages
-noinform	Disable informational messages
-rmic "options"	Set additional options to use for RMIC
-trace	Trace progress of the deploy tool
-sqlj	Use SQLJ instead of JDBC
-OCCColumn	Add a column for collision detection for WebSphere 6.0 or later release
-outer	Use OUTER semantics for path expressions in EQL queries for J2EE 1.3 applications
-complianceLevel	JDK level for compiler compliance

For a complete description of the EJBDeploy command and its parameters, see the Information Center. Search for *ejbdeploy*.

Example 14-5 shows a sample EJBDeploy run using the Plants by WebSphere EAR file.

Example 14-5 EJBDeploy sample run

```
C:\WebSphere\AppServer\bin>ejbdeploy c:\PlantsByWebSphereEnhanced.ear c:\temp
c:\PlantsByWebSphereEnhanced_Deployed.ear -dbvendor DB2UDB_V82
Starting workbench.
framework search path: c:\WebSphere\AppServer\deploytool\itp\plugins
Creating the project.
Deploying jar PlantsByWebSphereEJB
Validating
Generating deployment code
Generating DDL
Generating DDL
...
...
Invoking RMIC.
Writing output file
Shutting down workbench.
EJBDeploy complete.
0 Errors, 62 Warnings, 0 Informational Messages
```

Not shown in the listing above are the warnings that the validators spot. These warnings are things like if a local variable has been declared but is not used or if a Serializable class does not specify a static final long serialVersionUID. These warnings are not critical.

Tip: WebSphere Application Server also provides a set of Ant tasks that you can use to automate the packaging and deployment of your applications. One of those tasks allows you to call EJBDeploy. Search for *Ant tasks* in the Information Center for more details.

14.3 Deploying the application

In this section, we show the steps required to deploy the application to WebSphere Application Server. We show you how to deploy a regular EAR file as well as an Enhanced EAR file, and then also how to not honor the configuration information packaged into the Enhanced EAR file.

Follow these steps to deploy the application:

1. Select **Applications** → **Install New Application** from the administrative console navigation bar.
2. Check the **Local file system** box and click the **Browse** button to locate the `PlantsbyWebSphere.ear` file.

From the install windows, you can install files that are located either on the same machine as the browser you are using to access the WebSphere administrative console, the local file system option, or on the WebSphere Application Server itself, the remote file system option. If you select the Local file system option, the administrative console automatically uploads the file you select to the application server, or to the deployment manager if this is a distributed server environment. If you select the Remote file system check box, you can browse all the nodes in the cell to find the file. The file is then, if necessary, uploaded to the application server or deployment manager.

New in V6.1: WebSphere Application Server V6.1 allows you to take a shortcut when installing an application. If you select the **Prompt me only when additional information is required** option, only the windows where you actually need to fill out some information during installation are shown.

For this example, however, we will explain the options, so select **Show me all installation options and parameters**. Then click **Next**.

3. In the next window, specify default bindings for the application you are deploying. Unless you check the **Override** option, bindings already specified

in the EAR are not altered. The various bindings you can specify in this page are documented in Table 14-1 on page 914. If you do choose to override bindings, select **Generate Default Bindings** at the top of this window to apply changes to the application you are deploying.

In this example, the bindings were set in the application EAR file using the Application Server Toolkit and there is no need to override them. The defaults are also correct.

Table 14-1 Application default bindings

Binding name	Detailed information
EJB prefix	You can generate default EJB JNDI names using a common prefix. EJBs for which you did not specify a JNDI name will get a default name, built by concatenating the prefix and the EJB name. If you specify a prefix of myApp/ejb, then JNDI names default to myApp/ejb/EJBName, such as myApp/ejb/Account.
Override	Enter whether you want to override the current bindings. By default, existing bindings are not altered.
EJB 1.1 CMP bindings	You can bind all EJB 1.1 CMP entity beans to a specific data source, including user ID and password.
Connection Factory bindings	You can bind all EJB modules to a specific data source. You will have to go to the next window to override this setting at the EJB level.
Virtual host bindings	You can bind all Web modules to a specific virtual host, such as plantsbywebsphere_host.
Specify bindings file	You can also create a specific bindings file using your favorite editor and load it during application installation by clicking Browse next to the specific bindings file. For information about using a bindings file, see 14.3.1, "Using a bindings file" on page 919.

4. Click **Next**.

The rest of the wizard is divided into steps. The number of steps depends on your application, for example, if it contains EJB modules or Web modules, you will see windows prompting for the information necessary to deploy them.

5. Step 1: Select installation options.

Step 1 gives you a chance to review the installation options. You can specify various deployment options, such as JSP precompiling, and whether you want to generate EJB deployment code.

If you are deploying an Enhanced EAR file, this is where you make the decision whether to use the resource configuration information packaged in

the Enhanced EAR file or not. If the EAR file you are installing is an Enhanced EAR, the install window preselects the **Process embedded configuration** check box. If you do not want to use the resource configuration information packaged in the Enhanced EAR file, you must deselect this check box.

Selecting the Pre-compile JSP option makes WebSphere compile all JSPs in the EAR file during install time. This causes the time-consuming task of JSP compilation to be performed during install time instead of during run time, preventing the first user that accesses the application to pay that penalty.

A second alternative to pre-compile JSPs is to use the JspBatchCompiler script found in the bin directory of the profile you are using to compile the JSPs after the application has been installed.

This page also allows you to specify file permissions for files in your application. To use one of the predefined file permissions, select it, and then click **Set file permissions**. You can also specify your own file permissions using regular expressions.

New in V6.1: The administrative console displays the Application Build ID of the application being installed. This string is specified in the MANIFEST.MF file in the EAR file's META-INF folder and can be set using the Application Server Toolkit.

The following is an example of a version number:

Implementation-Version: Version 1.2.3

New in V6.1: The Remote Request Dispatcher is an extension to the Web container that allows frameworks, servlets, and JSPs to include content from outside of the current executing resource's JVM as part of the response sent to the client.

To enable this feature, select the corresponding check boxes to allow dispatching or servicing includes to/from remote resources. For more information about this feature, search the InfoCenter for Remote request dispatcher.

Click **Next**.

6. Step 2: Map modules to servers.

Select the server on which you want each module deployed. For better performance, we recommend that you deploy all modules from one application in a single server. Especially, do not separate the EJB clients, usually servlets in Web modules, from the EJBs themselves.

Click the  icon to select all modules in the Plants by WebSphere EAR file. In the Clusters and Servers box, select **PlantsByWebSphereServer**. Then click **Apply**. This assigns all modules to the PlantsByWebSphereServer application server. If you deploy to a cluster, select the cluster instead of the single application server.

See Figure 14-9 on page 916.

Web servers: If you have a Web server defined, select both the Web server and WebSphereBankServer in the server list. Press and hold the CTRL key to select multiple servers. Mapping Web modules to Web servers ensures the Web server plug-in will be generated properly.

Map modules to servers

Specify targets such as application servers or clusters of application servers where you want to install the modules that are contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that serve as routers for requests to this application. The plug-in configuration file (plugin-cfg.xml) for each Web server is generated, based on the applications that are routed through.

Clusters and Servers:

WebSphere:cell=kcgg1d8Cell,node=kcgg1d8Node,server=server1
 WebSphere:cell=kcgg1d8Cell,node=kcgg1d8Node,server=PlantsByWebSphereServer
 WebSphere:cell=kcgg1d8Cell,node=kcgg1d8Node,server=webserver1

Apply

Select	Module	URI	Server
☐	PlantsByWebSphere EJB Module	PlantsByWebSphereEJB.jar,META-INF/ejb-jar.xml	WebSphere:cell=kcgg1d8Cell,node=kcgg1d8Node,server=PlantsByWebSphereServer
☐	PlantsByWebSphereWeb	PlantsByWebSphereWeb.war,WEB-INF/web.xml	WebSphere:cell=kcgg1d8Cell,node=kcgg1d8Node,server=PlantsByWebSphereServer

Figure 14-9 Mapping modules to application servers

Note: Steps 3 - 11 allow you to define bindings. We have already taken care of this using the Application Server Toolkit when we packaged the EAR file. You can skip directly to Step 11 if you like. See 15 on page 918.

7. Step 3: Select current back-end ID.

A single EAR file can contain multiple database mappings. At deployment time, you can choose which one you want to use. In this case, set it to DB2UDBNT_V82_1, because this is the version we are using.

Click **Next**.

8. Step 4: Provide JSP reloading options for Web modules.

Allows you to configure if and how often WebSphere should check for updates to JSP files, and if they should be reloaded or not. In a production environment, you may want to disable this to improve performance.

Click **Next**.

9. Step 5: Map shared libraries.

If your application depends on shared libraries, you can specify them here. For more information about using shared libraries, see “Shared libraries” on page 809.

Click **Next**.

10. Step 6: Provide JNDI names for beans.

Use this window to bind the enterprise beans in your application or module to a JNDI name. In 13.3.1, “Defining EJB JNDI names” on page 842, we defined these values in Table 13-1, so the defaults shown should be correct.

Click **Next**.

11. Step 7: Map EJB references to beans.

Each EJB reference defined in your application must be mapped to an enterprise bean. We used the Application Server Toolkit to do this in 13.3.2, “Binding EJB and resource references” on page 844.

Click **Next**.

12. Step 8: Map default data source mapping for modules containing 2.x entity beans.

Specify the default data source for the EJB 2.x module containing 2.x CMP beans. In 13.3.3, “Defining data sources for entity beans” on page 846, we defined the JNDI name for the EJBs in the PlantsByWebSphere EJB module as eis/jdbc/PlantsByWebSphereDataSource_CMP. You see this in the window.

Click **Next**.

13. Step 9: Map data sources for all 2.x CMP beans.

Specify an optional data source for each 2.x CMP bean. Mapping a specific data source to a CMP bean overrides the default data source for the module containing the enterprise bean (defined in step 8, (12 on page 917). We do not need to do anything here.

Click **Next**.

14. Step 10: Map resource references to resources.

Each resource reference defined in the application must be mapped to the corresponding resource. The PlantsByWebSphere EJB module has several resource references for data sources, which are shown in this window.

Click **Next**.

At this step, we now get an Application Resource Warning. What this tells us is that a resource referenced by the application, mail/PlantsByWebSphere, is not defined for the scope to which we are installing our application. In fact, it is

not defined at all in our environment because we will not send any e-mails from the Plants by WebSphere. If you would like to do that, you should define a Mail session with the properties for your mail server and assign it a JNDI name of mail/PlantsByWebSphere.

Click **Continue**.

15. Step 11: Map virtual hosts for Web modules.

For each Web module, select the virtual host we created for the application (plantsbywebsphere_host).

Click **Next**.

16. Step 12: Map context roots for Web modules.

For each Web module, select the context root to bind the module against.

Click **Next**.

17. Step 13: Map security roles to users and groups.

Because the Plants by WebSphere EAR file contains security roles, we need to map them to users and groups in our target environment. However, because we have not enabled application security, WebSphere will not authenticate users trying to access the application. As a result, we do not need to map the roles to users and groups.

Click **Next**.

18. Step 14: Ensure all unprotected 2.x methods have the correct level of protection.

By default, EJB methods are unprotected. On this window, you can elect to refuse all calls to unprotected methods, or specify which methods you want to exclude.

Again, because we have not enabled J2EE security, WebSphere will not authenticate users trying to access the EJBs.

Click **Next**.

19. Step 15: Summary.

The Summary window gives an overview of application deployment settings. If those settings are fine, click **Finish** to deploy the application.

20. Save the configuration.

If you are working in a distributed server environment, make sure you synchronize the changes with the nodes so the application is propagated to the target application server (s).

21. If you mapped the Web modules to a Web server, make sure the Web server plug-in is regenerated and propagated to the Web server. For a quick refresh, restart the Web server.

Deployment is now complete. You can now launch the Plants by WebSphere application by pointing your browser to:

<http://www.plantsbywebsphere.com:9081/PlantsByWebSphere>

Make sure that the host name is one of the names you added to the `plantsbywebsphere_host` definition. See 14.1.5, “Creating the virtual host for IBM HTTP Server and Apache” on page 902.

Because the DB2 PLANTS database is now empty, it must be populated with data before the application will work. An administrative servlet is supplied for that purpose.

1. Click the **HELP** link in the upper right corner of the page, or go directly to <http://www.plantsbywebsphere.com:9081/PlantsByWebSphere/help.jsp>.
2. Select the **Logging** option and click **Save Setting**. This enables log messages in the application.
3. Click the **(Re)-populate database** link. This will load images and product descriptions from the EAR file and store them in the database.

After the database has been populated, you can test the application.

14.3.1 Using a bindings file

If generating default bindings during deployment, default names suitable for most applications are used. However, these defaults do not work if:

- ▶ You want to explicitly control the global JNDI names of one or more EJB files.
- ▶ You need tighter control of data source bindings for container-managed persistence (CMP) beans. That is, you have multiple data sources and need more than one global data source.
- ▶ You must map resource references to global resource JNDI names that are different from the `java:comp/env` name.

In such cases, you can use a specific bindings file to customize the bindings created.

To use a bindings file when installing an application, load it by clicking **Browse** next to the Specific bindings file option. When using this file, only specify bindings that differ from the defaults, not the full bindings.

Example 14-6 is an example showing a bindings file used to change the JNDI name of an EJB.

Example 14-6 Using a bindings file to change the JNDI of an EJB

```
<?xml version="1.0"?>
<!DOCTYPE dfldbndngs SYSTEM "dfldbndngs.dtd">
<dfldbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>helloEjb.jar</jar-name>
      <!-- this name must match the module name in the .ear file -->
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>HelloEjb</ejb-name>
          <!-- this must match the <ejb-name> entry in the EJB jar DD -->
          <jndi-name>com/acme/ejb/HelloHome</jndi-name>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfldbndngs>
```

14.4 Deploying application clients

To run a Java-based client/server application, the client application executes in a client container of some kind. You might, for example, use a graphical Swing application that calls EJBs on an application server. WebSphere Application Server V6 supports the following five types of application client environments:

- ▶ J2EE application client

This client uses services provided by the J2EE client container.

This client is a Java application program that accesses EJBs, JDBC databases, and JMS queues. The J2EE application client program runs on client machines. This program allows the same Java programming model as other Java programs. However, the J2EE application client depends on the application client run time to configure its execution environment, and it uses the JNDI name space to access resources, the same as you would in a normal server application (like a servlet).

The J2EE application client brings the J2EE programming model to the client, and provides:

- XML deployment descriptors

- J2EE naming (java:comp/env), including EJB references and resource references

The J2EE application client is launched using the launchClient script, which sets up the environment with the necessary classpaths, and so on, for you.

- ▶ Thin application client

This client does not use services provided by the J2EE client container.

This client provides a lightweight Java client programming model and is best suited for use in situations where a Java client application exists, but the application must be enhanced to make use of EJBs. It can also be used where the client application requires a thinner, more lightweight environment than the one offered by the J2EE application client. The Thin application client includes the IBM JDK. When launching the Thin application client, you must set up the correct classpaths yourself and make sure that the required libraries for your application and the WebSphere libraries are included.

- ▶ Pluggable application client

This client does not use services provided by the J2EE Client Container.

This client is similar to the Thin application client, but does not include a JVM. The user is required to provide a JVM. It can also use the Sun JDK instead of the IBM JDK.

- ▶ Applet application client

In the Applet client model, a Java applet embedded in an HTML document executes in a Web browser. With this type of client, the user accesses an enterprise bean in the application server through the Java applet in the HTML document.

- ▶ ActiveX® to EJB Bridge application client

The ActiveX application client allows ActiveX programs to access enterprise beans through a set of ActiveX automation objects. The ActiveX application client uses the Java Native Interface (JNI) architecture to programmatically access the Java virtual machine (JVM) API. Therefore, the JVM code exists in the same process space as the ActiveX application (Visual Basic®, VBScript, or Active Server Pages files) and remains attached to the process until that process terminates.

The capabilities of the different application clients are shown in Table 14-2.

Table 14-2 Application client features comparison

Available functions	J2EE client	Thin client	Pluggable client	Applet client	ActiveX client
Provides all the benefits of a J2EE platform.	Yes	No	No	No	Yes
Portable across all J2EE platforms.	Yes	No	No	No	No
Provides the necessary run time support for communication between a client and a server.	Yes	Yes	Yes	Yes	Yes
Supports the use of nicknames in the deployment descriptor files.	Yes	No	No	No	Yes
Supports use of the RMI-IIOP protocol.	Yes	Yes	Yes	Yes	Yes
Browser-based application.	No	No	No	Yes	No
Enables development of client applications that can access enterprise bean references and CORBA object references.	Yes	Yes	Yes	Yes	Yes
Enables the initialization of the client application run time environment.	Yes	No	No	No	Yes
Supports security authentication to enterprise beans.	Yes	Yes	Yes	Limited	Yes
Supports security authentication to local resources.	Yes	No	No	No	Yes
Requires distribution of application to client machines.	Yes	Yes	Yes	No	Yes

Available functions	J2EE client	Thin client	Pluggable client	Applet client	ActiveX client
Enables access to enterprise beans and other Java classes through Visual Basic, VBScript, and Active Server Pages (ASP) code.	No	No	No	No	Yes
Provides a lightweight client suitable for download.	No	Yes	Yes	Yes	No
Enables access JNDI APIs for enterprise bean resolution.	Yes	Yes	Yes	Yes	Yes
Runs on client machines that use the Sun Java Runtime Environment.	No	No	Yes	No	No
Supports CORBA services (using CORBA services can render the application client code nonportable).	Yes	No	No	No	No

Install the application client environments from the WebSphere installation windows by selecting the **Launch the installation wizard for WebSphere Application Clients** option. The installation package contains the following installable components:

- ▶ IBM Java Runtime Environment (JRE™), or an optional full Software Development Kit
- ▶ WebSphere Application Server run time for J2EE application client applications, or Thin application client applications
- ▶ An ActiveX to EJB Bridge run time for ActiveX to EJB Bridge application client applications (only for Windows)
- ▶ IBM plug-in for Java platforms for Applet client applications (Windows only)

Note: The J2EE client is automatically installed as part of a full WebSphere install. In other words, if you will run the client application on a machine that already has WebSphere installed, you do not need to install the WebSphere J2EE client on top.

14.4.1 Defining application client bindings

The Plants by WebSphere sample application does not provide any client application. For the purpose of discussing the client container, we will therefore use the WebSphere Bank sample application that was shipped with WebSphere Application Server V6.0 but has been removed in V6.1.

WebSphere Bank provides four client applications: the GetAccounts, FindAccounts, TransferWS and TransferJMS clients. The various client applications demonstrate the capabilities of the WebSphere Bank sample application.

For an application client to be able to access resources, such as EJBs provided by a J2EE server application, the proper bindings must be set up. You need to specify the complete naming structure to reach the server where the EJBs are deployed. For example, the machine where we deployed the application for testing has the Network Deployment version installed. The WebSphere Bank application is running in the WebSphereBank application server on node ITSONode1.

Therefore, the ejb/Bank/Customer EJB reference can be bound to:

cell/nodes/ITSONode1/servers/WebSphereBankServer/ejb/Bank/Customer

If you have created a cluster of application servers, use:

cell/clusters/<clusterName>/ejb/Bank/Customer

You will also need to change the provider URL, according to the target server. If you are running in a single server environment you can simply use:

ejb/Bank/Customer

When you have configured the proper bindings for the resources, you must export the EAR file and copy it to the client machine. Although you do not need the complete contents of the EAR file to run the application client, for example, the Web modules, it is better to keep a single EAR file. This is mainly for maintenance purposes.

14.4.2 Launching the J2EE client

A J2EE client application needs a container to run in. In this example, we will use the J2EE application client container. This container can be started using the launchClient program in the <<was_home>>/bin directory. The launchClient program has the following syntax:

```
Usage: launchClient [-profileName pName | -JVMOptions options | -help | -?]  
<userapp> [-CC<name>=<value>] [app args]
```

The elements of syntax are:

-profileName	This option defines the profile of the Application Server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment or in an Application Clients installation. The default is default_profile.
-JVMOptions	This is a valid Java standard or nonstandard option string. Insert quotation marks around the option string.
-help, -?	Print the usage information.
<userapp.ear>	Type the path/name of the .ear file containing the client application.

The -CC properties are for use by the Application Client run time. There are numerous parameters available and because of this we only describe the more commonly used ones. For full explanation of all parameters, execute **launchClient -help**.

-CCverbose	Use this option with <true false> to display additional informational messages. The default is false.
-CCclasspath	This property is a classpath value. When an application is launched, the system classpath is not used. If you need to access classes that are not in the EAR file or part of the resource classpaths, specify the appropriate classpath here. Multiple paths can be concatenated.
-CCjar	This is the name of the client JAR file within the EAR file that contains the application you want to launch. This argument is only necessary when you have multiple client JAR files in the EAR file.
-CCBootstrapHost	This option is the name of the host server you want to connect to initially. The format is your.server.ofchoice.com.
-CCBootstrapPort	This option is the server port number. If not specified, the WebSphere default value (2809) is used.

- CCproviderURL** This option provides bootstrap server information that the initial context factory can use to obtain an initial context. WebSphere Application Server initial context factory can use either a CORBA object URL or an IIOP URL. CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. This value can contain more than one bootstrap server address. This feature can be used when attempting to obtain an initial context from a server cluster. In the URL, you can specify bootstrap server addresses for all servers in the cluster. The operation will succeed if at least one of the servers is running, eliminating a single point of failure. The address list does not process in a particular order. For naming operations, this value overrides the -CCBootstrapHost and -CCBootstrapPort parameters. An example of a CORBA object URL specifying multiple systems is:
-CCproviderURL=corbaloc:iiop:myserver.mycompany.com:9810,:mybackupserver.mycompany.com:2809
- CCtrace** Use this option with <true|false> to have WebSphere write debug trace information to a file. The value true is equivalent to a trace string value of com.*=all=enabled. Instead of the value true you can specify a trace string, for example, -CCtrace=com.ibm.ws.client.*=all=enabled. Multiple trace strings can be specified by separating them with a colon (:). You might need this information when reporting a problem to IBM Service. The default is false.
- CCtracefile** This option is the name of the file to which to write trace information. The default is to output to the console.
- CCpropfile** This option is the name of a properties file containing launchClient properties. In the file, specify the properties without the -CC prefix. For example: verbose=true.

The app args are for use by the client application and are ignored by WebSphere.

To start the WebSphere Bank GetAccounts client using the launchClient command, execute the command shown in Figure 14-7.

Example 14-7 Launching WebSphere Bank application client

```
C:\WebSphere\AppServer\profiles\AppSrv01\bin>launchClient.bat
c:\WebSphereBankClient.ear -CCBootstrapPort=2809 -CCjar=GetAccounts.jar 100
```

```
IBM WebSphere Application Server, Release 6.1
J2EE Application Client Tool
Copyright IBM Corp., 1997-2006
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has
completed.
WSCL0014I: Invoking the Application Client class
com.ibm.websphere.samples.bank.client.GetAccounts
Get the account numbers owned by a certain customer

Getting the customer home...

Done....

Finding the customer....
Done....
Account Number: 101
All done!

Finding the customer from invoke ....
Account Number: 101
All done!

Get accounts owned by customer from invoke...
Account Number: 101
Account balance: 200.0
Get number of customer accounts from invoke...
Account :1
All done!
```

Because the WebSphereBank EAR file contains multiple client applications (JAR files), the -CCjar option must be used to specify which client application to launch.

The WebSphere Bank GetAccounts client application is a text-mode application that simply displays the accounts for a certain customer number 100 in our example.

14.5 Updating applications

WebSphere Application Server has features that allow applications to be updated and restarted at a fine-grained level. It is possible to update only parts of an application or module and only the necessary parts are restarted. You can:

- ▶ Replace an entire application (.ear file).
- ▶ Replace, add, or remove a single module (.war, EJB .jar, or connector .rar file).
- ▶ Replace, add, or remove a single file.
- ▶ Replace, add and remove multiple files by uploading a compressed file describing the actions to take.

If the application is running while being updated, WebSphere Application Server automatically stops the application, or only its affected components, updates the application, and restarts the application or components.

When updating an application, only the portion of the application code that changed needs to be presented to the system. The application management logic calculates the minimum actions that the system needs to execute in order to update the application. Under certain circumstances, the update can occur without stopping any portion of the running application.

WebSphere Application Server also has support for managing applications in a cluster for continuous availability. The action, Rollout Update, sequentially updates an application installed on multiple cluster members across a cluster. After you update an application's files or configuration, use the Rollout Update option to install the application's updated files or configuration on all cluster members of a cluster on which the application is installed.

Rollout Update does the following for each cluster member in sequence:

1. Saves the updated application configuration.
2. Stops all cluster members on a given node.
3. Updates the application on the node by synchronizing the configuration.
4. Restarts the stopped cluster members on that node.

This action updates an application on multiple cluster members while providing continuous availability of the application.

14.5.1 Replacing an entire application EAR file

To replace a full EAR with a newer version, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.

2. On the Preparing for the application installation window, select the **Replace the entire application** option.
3. Select either the **Local file system** or **Remote file system** option. Click the **Browse** button to select the updated EAR file. Click **Next**.
4. Proceed through the remaining windows and make any changes necessary. For information about the windows, see “Deploying the application” on page 913. On the Summary window, click **Finish**.
5. When the application has been updated in the Master repository, select the **Save** link.
6. If you are working in a distributed server environment, make sure that you also synchronize the changes with the nodes.
7. If the application update changes the set of URLs handled by the application (servlet mappings added, removed, or modified), make sure the Web server plug-in is regenerated and propagated to the Web server.

Note: It may take a few seconds for the WebSphere run time to pick up the changes and restart the application as necessary. If your changes do not seem to have effect, wait and try again. You can also look at the SystemOut.log file for the application server to see when it has restarted the application.

14.5.2 Replacing or adding an application module

To replace only a module, such as an EJB or Web module of an application, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.
2. On the Preparing for the application installation window, select the **Replace or add a single module** option.
3. In the Specify the path beginning with the installed application archive file... field, enter the relative path to the module to replace. For example, if you were to replace the HelloWorld module, enter HelloWorld. If you enter a path or file that does not exist in the EAR file, it will be added.
4. Select either the **Local file system** or **Remote file system** option and click the **Browse** button to select the updated module.
5. For Web modules, also enter the context root (for example, HelloWorld) in the Context root field.
6. Click **Next**.

7. Proceed through the remaining windows and make any necessary changes. For information about the windows, see “Deploying the application” on page 913. On the Summary window, click **Finish**.

Note: If you are adding a module, make sure to select the correct target server for the module in the Map modules to servers step.

8. When the application has been updated in the Master repository, select the **Save** link.
9. If you are working in a distributed server environment, make sure that you also synchronize the changes with the nodes.
10. If the application update changes the set of URLs handled by the application (servlet mappings added, removed, or modified), make sure the Web server plug-in is regenerated and propagated to the Web server.

Note: Modules can also be managed using the Manage Modules page. Select **Applications** → **Enterprise Applications** and click the link for the application. Then click the **Manage Modules** link in the Modules section. Select the module to modify and then click the **Remove**, **Update**, or **Remove File** buttons.

14.5.3 Replacing or adding single files in an application or module

To replace a single file, such as a GIF image or a properties file in an application or module, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.
2. On the Preparing for the application installation window, select the **Replace or add a single file** option.
3. In the **Relative path to file** field, enter the relative path to the file to replace in the EAR file. For example, if you were to replace the logo.gif in the images directory of the HelloWorld.war Web module, you would enter HelloWorld.war/images/logo.gif. If you enter a path or file that does not exist in the EAR file, it will be added.
4. Select either the **Local file system** or **Remote file system** option and click the **Browse** button to locate the updated file. Click **Next**.
5. On the Updating Application window, click **OK**.
6. When the application has been updated in the Master repository, select the **Save** link.

7. If you are working in a distributed server environment, make sure that you also synchronize the changes with the nodes.

14.5.4 Removing application content

Files can also easily be removed either from an EAR file or from a module in an EAR file.

Removing files from an EAR file

To remove a file from an EAR file, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to remove the file from and click the **Remove File** button.
2. In the Remove file dialog box, select the file to be removed and click **OK**.
3. Save the configuration.

Removing files from a module

To remove a file from a module, do the following:

1. Select **Applications** → **Enterprise Applications** and click the link for the application to which the module belongs.
2. Click the **Manage Modules** link under the Modules section.
3. Select the module to remove the file from and click the **Remove File** button.
4. In the Remove a file from a module dialog, select the file to be removed and click **OK**.
5. Save the configuration.

14.5.5 Performing multiple updates to an application or module

Multiple updates to an application and its modules can be packaged in a compressed file, .zip, or .gzip format, and uploaded to WebSphere Application Server. The uploaded file is analyzed and the necessary actions to update the application are taken.

Depending on the contents of the compressed file, this method to update an application can replace files in, add new files to, and delete files from the installed application all in one single administrative action. Each entry in the compressed file is treated as a single file, and the path of the file from the root of the compressed file is treated as the relative path of the file in the installed application.

- To replace a file, a file in the compressed file must have the same relative path as the file to be updated in the installed application.

- ▶ To add a new file to the installed application, a file in the compressed file must have a different relative path than the files in the installed application.
- ▶ To remove a file from the installed application, specify metadata in the compressed file using a file named META-INF/ibm-partialapp-delete.props at any archive scope. The ibm-partialapp-delete.props file must be an ASCII file that lists files to be deleted in that archive with one entry for each line. The entry can contain a string pattern, such as a regular expression that identifies multiple files. The file paths for the files to be deleted must be relative to the archive path that has the META-INF/ibm-partialapp-delete.props file.
- ▶ To delete a file from the EAR file (not a module), include a META-INF/ibm-partialapp-delete.props file in the root of the compressed file. In the .props file, list the files to be deleted. File paths are relative to the root of the EAR file.

For example, to delete a file named docs/readme.txt from the root of the HelloApp.ear file, include the line docs/readme.txt in the META-INF/ibm-partialapp-delete.props file in the compressed file.

- ▶ To delete a file from a module in the EAR, include a module_uri/META-INF/ibm-partialapp-delete.props file in the compressed file. The module_uri part is the name of the module, such as HelloWeb.war.

For example, to delete images/logo.gif from the HelloWeb.war module, include the line images/logo.gif in the HelloWeb.war/META-INF/ibm-partialapp-delete.props file in the compressed file.

- ▶ Multiple files can be deleted by specifying each file on its own line in the metadata .props file.

Regular expressions can also be used to target multiple files. For example, to delete all JavaServer Pages (.jsp files) from the HelloWeb.war file, include the line `.*jsp` in the HelloWeb.war/META-INF/ibm-partialapp-delete.props file. The line uses a regular expression, `.*jsp`, to identify all .jsp files in the HelloWeb.war module.

As an example, assume we have prepared the compressed HelloApp_update.zip file shown in Figure 14-10.

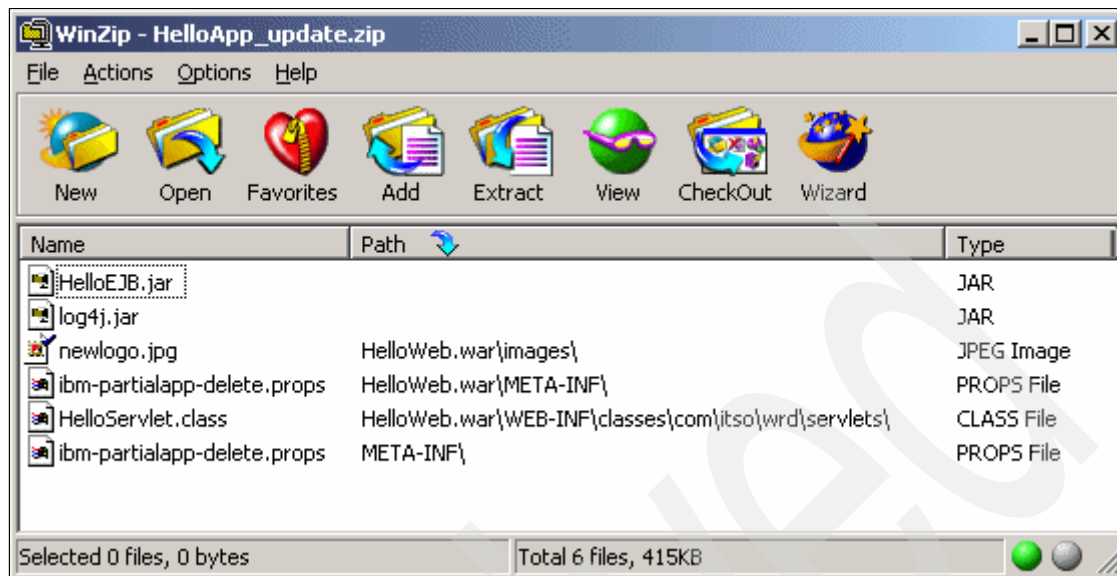


Figure 14-10 HelloApp_update.zip compressed file

The META-INF/ibm-partialapp-delete.props file contains the following line:

docs/readme.txt

The HelloWeb.war/META-INF/ibm-partialapp-delete.props contains the following lines:

images/logo.gif

When performing the partial application update using the compressed file, WebSphere does the following:

- ▶ Adds the log4j.jar file to the root of the EAR.
- ▶ Updates the entire HelloEJB.jar module.
- ▶ Deletes the docs/readme.txt file (if it exists) from the EAR file, but not from any modules.
- ▶ Adds the images/newlogo.jpg file to the HelloWeb.war module.
- ▶ Updates the HelloServlet.class file in the WEB-INF/classes/com/itso/wrd/servlets directory of the HelloWeb.war module.
- ▶ Deletes the images/logo.gif file from the HelloWeb.war module.

To perform the actions specified in the HelloWorld_updated.zip file, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.
2. On the Preparing for the application installation window, select the **Partial Application** option.
3. Select either the **Local file system** or **Remote file system** option and click the **Browse** button to select the compressed ZIP file with the modifications you have created. Click **Next**.
4. On the Updating Application window, click **OK**.
5. When the application has been updated in the Master repository, select the **Save To Master Configuration** link.
6. If in a distributed server environment, make sure the **Synchronize changes with Nodes** option is selected so that the application is distributed to all nodes. Click the **Save** button. The application is distributed to the nodes, updated, and restarted as necessary.
7. If the application update changes the set of URLs handled by the application (servlet mappings added, removed or modified), make sure the Web server plug-in is regenerated and propagated to the Web server.

14.5.6 Rolling out application updates to a cluster

The new Rollout Update feature allows you to easily roll out a new version of an application, or part of an application using the techniques described previously, to a cluster. The Rollout Update feature takes care of stopping the cluster members, distributing the new application, synchronizing the configuration, and restarting the cluster members. The operation is done sequentially over all cluster members in order to keep the application continuously available.

When stopping and starting the cluster members, the Rollout Update feature works on node level, so all cluster members on a node are stopped, updated, and then restarted, before the process continues to the next node.

Because the Web server plug-in module is not able to detect that an individual application on an application server is unavailable, the Rollout Update feature always restarts the whole application server hosting the application. Because of this, if HTTP session data is critical to your application, it should either be persisted to database or replicated to other cluster members using the memory-to-memory replication feature.

The order in which the nodes are processed and the cluster members are restarted is the order in which they are read from the cell configuration repository.

There is no way to tell the Rollout Update feature to process the nodes and cluster members in any particular order.

Assume we have an environment with two nodes, ITSONode1 and ITSONode2, and a cluster called HelloCluster, which has one cluster member on each node (HelloServer1 on ITSONode1 and HelloServer2 on ITSONode2). Assume we have an application called HelloApp deployed and running on the cluster. To update this application using the Rollout Update feature we would do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.
2. On the Preparing for the application installation window, select the appropriate action depending on the type of update. In this example, we will update the entire application EAR to a new version, so we select the **Replace the entire application** option.
3. Select either the **Local file system** or **Remote file system** option and click the **Browse** button to select the updated EAR file. Click **Next**.
4. Proceed through the remaining windows and make any changes necessary. For information about the windows, see “Deploying the application” on page 913. On the Summary window, click **Finish**.

5. When the application has been updated in the master repository, the status window shown in Figure 14-11 on page 936 is displayed.

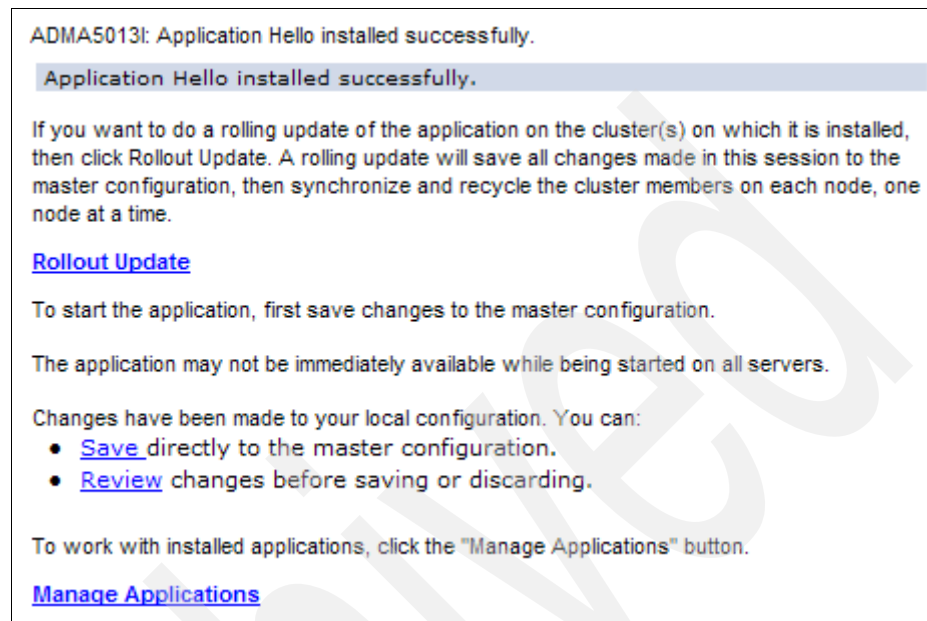


Figure 14-11 Preparing for application rollout

You then have two options to start the rollout action:

- Click the **Rollout Update** link.
- Click the **Manage Applications** link and on the Enterprise Applications window, select the application and click the **Rollout Update** button.

Note: Do not click the Save to Master Configuration link or otherwise save the configuration yourself. The Rollout Update will do that for you. If you save the configuration yourself, the rollout update action will be canceled and it will be handled as a normal application update.

During the rollout, the window in Figure 14-12 on page 937 is displayed in the status window.

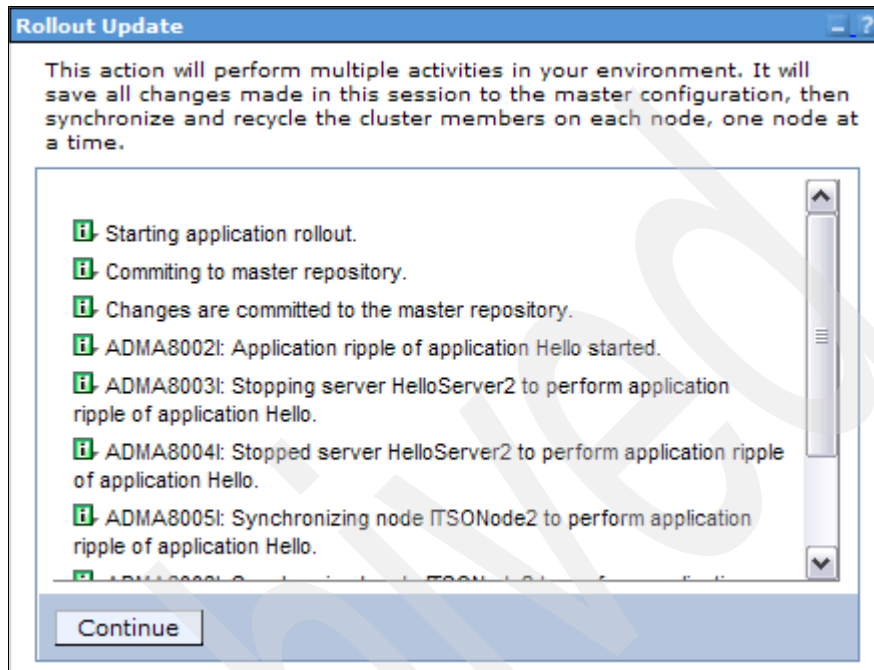


Figure 14-12 Rolling out an application

For each node, the cluster members are stopped, the application is distributed, and they are restarted. When the rollout has completed (the last message says “The application rollout succeeded”, click **Continue**.

6. If the application update changes the set of URLs handled by the application (servlet mappings added, removed, or modified), make sure the Web server plug-in is regenerated and propagated to the Web server.

Note: The automatic file synchronization of the node agent is temporarily disabled during the rollout process and then re-enabled afterwards, if it was previously enabled. The Rollout Update feature works regardless of the automatic file synchronization setting. However, in production systems, the automatic synchronization is often disabled anyway to give the administrator greater control over exactly when changes made to the cell configuration are distributed to the nodes.

Although the Rollout Update feature makes it very easy to roll out an application to a cluster while keeping the application continuously available, make sure that your application can handle the roll out.

For example, assume you have version 1.0 of an application running in a cluster consisting of two application servers, server1 and server2, and that HTTP session data is persisted to a database. When you roll out version 2.0 of the application and server1 is stopped, the Web server plug-in redirects the users on server1 to server2. Then, when server1 is started again, bringing up version 2.0 of the application, the plug-in will start distributing requests to server1 again. Now, if the application update incurred a change in the interface of any class stored in the HTTP session, when server1 tries to get these session objects from the database, it might run into a deserialization or class cast exception, preventing the application from working properly.

Another situation to consider is when the database structure changes between application versions, as when tables or column names change name or content. In that case, the whole application might need to be stopped and the database migrated before the new version can be deployed. The Rollout Update feature would not be suitable in that kind of scenario.

So it is very important to understand the changes made to your application before rolling it out.

14.5.7 Hot deployment and dynamic reloading

Hot deployment and dynamic reloading characterize how application updates are handled when updates to the applications are made by directly manipulating the files on the server. In either case, updates do not require a server restart, though they might require an application restart:

- Hot deployment of new components

Hot deployment of new components is the process of adding new components, such as WAR files, EJB JAR files, EJBs, servlets, and JSP files to a running application server without having to stop and then restart the application server.

However, in most cases, such changes require the application itself to be restarted, so that the application server run time reloads the application and its changes.

► Dynamic reloading of existing components

Dynamic reloading of existing components is the ability to change an existing component without the need to restart the application server for the change to take effect. Dynamic reloading can involve changes to the:

- Implementation of an application component, such as changing the implementation of a servlet
- Settings of the application, such as changing the deployment descriptor for a Web module

To edit the files manually, locate the binaries in use by the server. See “Repository files used for application execution” on page 44. Although the application files can be manually edited on one or more of the nodes, these changes will be overwritten the next time the node synchronizes its configuration with the deployment manager. Therefore, we recommend that manual editing of an application’s files should only be performed in the master repository, located on the deployment manager machine.

Note: Unless you are familiar with updating applications by directly manipulating the server files, it might be better to use the administrative console Update wizard.

There are three settings that affect dynamic reload:

► Reload classes when application files are updated

In order for application files to be reloaded automatically after an update, Reload classes when application files are updated must be enabled and the Polling interval for updated files must be greater than 0.

Select **Applications** → **Enterprise Applications**, and click the link for the application. In the Detail properties section, click the **Class loading and update detection** link.

► Application Server class loader policy

The application server’s class loader policy should be set to Multiple. If it is set to Single, the application server will need to be restarted after an application update.

Select **Servers** → **Application Servers**, and click the **Server** link. The setting is found in the General Properties section.

► JSP Reload options for Web modules

A Web container reloads a Web module only when this setting is enabled.

Select **Applications** → **Enterprise Applications**, and click the link for the application. In the Web Module Properties section, click the **JSP reload**

options for web modules link, and then select the JSP enable class reloading option and enter a polling interval.

For more information about using hot deployment and dynamic reload, see the *Updating applications* and *Hot deployment and dynamic reloading* topics in the Information Center.